The Report committee for Callie Edward Muckelroy Certifies that this is the approved version of the following report:

Multichannel Digital Voltmeter

Using the LM3S8962 Evaluation Board

APPROVED BY

SUPERVISING COMMITTEE:

_____

Jonathan Valvano, Supervisor

_____

William Bard, Co-Supervisor

Multichannel Digital Voltmeter

Using the LM3S8962 Evaluation Board

by

Callie Edward Muckelroy B.S.

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2016

Multichannel Digital Voltmeter

Using the LM3S8962 Evaluation Board

by

Callie Edward Muckelroy M.S.E.

The University of Texas at Austin, 2016

SUPERVISORS: Jonathan Valvano, William Bard

This project is a proof of concept and prototype in both hardware and software, which demonstrates a useful and flexible 4-channel voltmeter using an LM3S8962 Evaluation Board, and commodity electronic components to serve as a shield circuit. The shield circuit protects the evaluation board from over/under voltage damage, and biases the incoming voltages to provide usable input voltages for the LM3S8962 ADC inputs.

The report begins with an explanation of the goals of the project, and research of what current products exist on the commercial market to meet the needs of multichannel voltage measurement. In the interest of cost savings, and in the interest of an academic exploration of the capabilities of the inexpensive evaluation boards on the market, the author explores how the ADC inputs of the LM3S8962 can serve some of the same needs of much more expensive commercial products in voltage measurement.

Several obstacles were encountered in the design and construction stages. A description of the obstacles, and how they were overcome is described.

The end design of both the shield circuit and software is then detailed end-to-end, along with an explanation of how to use the end product. Finally, the accuracy of the project is assessed and demonstrated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Introduction

Voltmeters are a vital tool to working with electric circuits. They measure the electric potential between two points in a circuit, transducing this potential into a device that makes a meaningful (usually numerical) output. There is sometimes a need to measure more than one voltage at a time. For example, it can be difficult, risky, or time consuming to physically move the probe to another location in the circuit. Or, it may be desired to compare two voltage points in a circuit in real time. One solution to this problem is to use two separate meters. So, what happens if one wishes to measure three, four, or more readings at once? At some point, one wonders if there is a more efficient way to take multiple readings at once, without having an absurd number of meters arranged in an awkward fashion. Commercial solutions are available to meet this need, but they usually cost in the thousands of dollars.[1]

Furthermore, most inexpensive digital voltmeters do not have a convenient way to calibrate the meter. Those that do have a way to calibrate the meter require access to the internal components of the meter. The calibration procedure is not always documented for the owner of the meter [2]. Few meters have ways to customize the way that the data is displayed, or provide ways to send the data to other devices or peripherals.

Microcontrollers can give all of the flexibility that traditional voltmeters lack. They provide a number of analog to digital converters (ADCs). The LM3S8962 evaluation board, in particular, has four ADC inputs [3 p.13]. It is based on the Cortex M3 architecture. As of this writing, many suppliers sell chips based on the Cortex M3 similar to the LM3S8962 chip for $5 USD or less. If a single microcontroller can take the place

of four voltmeters, this is a possibility well worth exploring. Before going down this road, a shield circuit had to be constructed to protect the evaluation board from damage. Then, code had to be written to translate the adapted ADC values from the shield circuit into actual voltages. Finally, all this data had to be displayed to the user in a meaningful way, also implementing auto-ranging for ease of use.

CHAPTER 1: DESCRIPTION OF SHIELD CIRCUIT

The ADC channels needed to measure from -60 to +60 Volts AC or DC. Six ranges are offered: +60V DC, +20V DC, ±60V DC ±20V DC, 60V AC and 20V AC by way of a MOSFET-switched voltage divider circuit. A separate board houses this voltage divider circuit, as well as a voltage clamping circuit, which protects the ADC and limits the measured voltage (V_{meas}) to a maximum of 3V. This separate circuit as a whole is referred to as the "shield circuit".

Note: For the purpose of readability, the remainder of this report and comments of the code will refer to the ADC channels as channels 1, 2, 3, and 4. The actual labels on the evaluation board are `ADC0`, `ADC1`, `ADC2` and `ADC3`. Likewise, the reading from the ADC FIFO is taken from `ADC_CTL_CH0`, `ADC_CTL_CH1`, `ADC_CTL_CH2` and `ADC_CTL_CH3` respectively.

Since the shield circuit must feed four ADC channels, the same circuit is multiplied 4 times. Refer to Figure 1 for the schematic diagram of channel 1 of the shield circuit. The only difference between the other channels is the GPIO pin assignments, and differences in where the ADC V_{meas} pins are connected to. See Appendix A for a full list of GPIO pin assignments for all 4 channels.

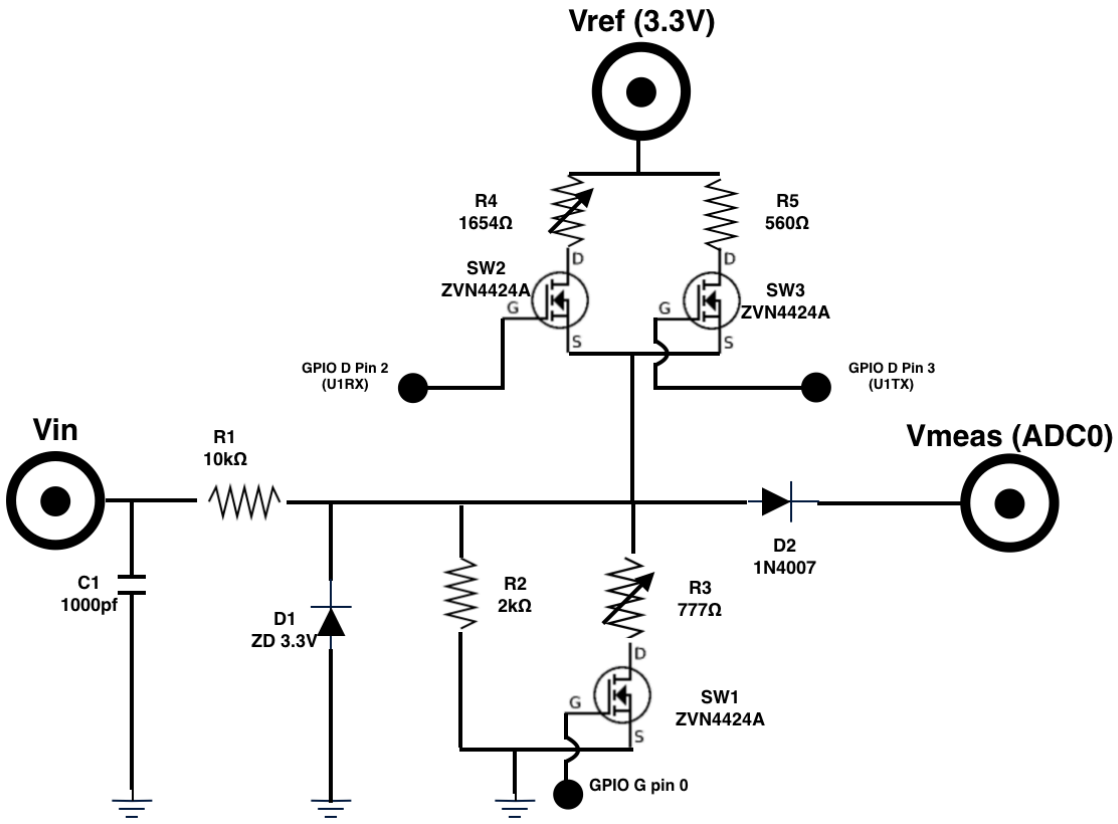


FIGURE 1: Channel 1 Shield Schematic

The shield circuit was inspired by a somewhat related project for Arduino, which is in the public domain [4]. The shield circuit is part voltage divider network, and part biasing. A 3.3V reference voltage from the LM3S8962 board is connected to the upper junction of R4 and R5. These resistors are taken in or out of the circuit by their own MOSFET switches of type ZVN4424A[6]. The gates of each of these MOSFETS are connected to dedicated GPIO pins of the LM3S8962. When high, these GPIO pins will send 3.3 V to the MOSFET gate. In other words, when the MOSFET gate is "high", the resistor that the MOSFET is attached to will be added to the circuit. When not "high", the connected resistor is effectively non-existent, and is in no way affecting the voltage being

measured. The RON (on resistance) of these MOSFETS is very low, at approximately 4.3 $\Omega_{[6\ p.2]}$.

In other words, when SW2 and SW3 are "OFF", there is no V_{ref} in the measurement, because the resistors connecting V_{ref} to the rest of the circuit are effectively disconnected from the rest of the circuit by way of the MOSFET switches.

The junction of R1, R2, R3, R4 and R5 forms a biased, divided voltage. This allows negative or positive voltages to be fed to V_{in}. In both cases, a non-negative voltage ranging between 0 and 3 arrives at the ADC at V_{meas}.

The voltage clamping zener diode cathode is also at this junction, and is used to limit the voltage, never to exceed 3V. The zener diode was carefully chosen so that even with a significant overload (up to 100V V_{in}), no more than 3V ever made its way to V_{meas}.

The LM3S8962 datasheet states that the absolute minimum voltage that may touch the ADC pins is $-0.3V_{[5\ p.748]}$. To protect V_{meas} from going negative, a 1N2007 diode is wired between the zener diode cathode and V_{meas}. Although the zener voltage for the zener diode is 3.3V, the surrounding components help to slightly pad the output.

This is a six-range meter in that two of these ranges are used for AC voltage calculations. The AC ranges use the same shield circuit states as the ±60V DC and ±20V DC ranges. Thus, the shield circuit is electronically in only one of four states. Table 1 describes the states of the shield circuit for each range, and the details after Table 1 explain these states.

SWITCH / **RANGE**	SW1	SW2	SW3
+60 DC	**ON**	OFF	OFF
+20 DC	OFF	OFF	OFF
±60V DC	**ON**	OFF	**ON**
±20V DC	OFF	**ON**	OFF
60V AC	**ON**	OFF	**ON**
20V AC	OFF	**ON**	OFF

TABLE 1: MOSFET State Table

+60V DC Range: When in this state, SW1 adds variable resistor R3 (777Ω) to the circuit so that it is in parallel with R2 (2kΩ) to ground. The two in parallel form a single resistor that is 560Ω. SW2 and SW3 are off, so that no V_{ref} bias voltage is present. This is the state that the shield circuit is in upon first power-up, and is most effective at measuring positive voltages up to +60V. It can safely tolerate negative voltages, but will not know what the negative voltages are without adding V_{ref} bias to the circuit via a different range.

+20V DC Range: When in this state, SW1 is off, meaning the only resistor of the pair to ground is R2 (2kΩ). This range is best for measuring smaller voltages not to exceed +20V, with higher resolution at that range. It can safely tolerate negative voltages, but will not know what those are without adding V_{ref} bias to the circuit via a different range.

±60V DC Range: When in this state, SW1 is on, as is SW3. With SW3 turned on, V_{ref} is mixed with the incoming V_{in}, applying a DC bias to the voltage, making possible a usable reading in the negative V_{in} range. When V_{in} drops below -60V, V_{meas} is at -3mV, and the microcontroller software will display a bottom-most value just below -60V. A high degree of resolution (approximately 0-2 V_{meas}) is afforded the negative Vin range. But, a much smaller degree of resolution (0.8 V_{meas}) is offered when measuring positive voltages in this range. This is the reason that positive-only ranges are still being offered, so that the meter can recognize an AC signal that alternates between the positive and negative voltages, and if necessary, switch to the +60 or +20 range to get an accurate reading of positive voltages

±20V DC Range: When in this state, SW2 is on, and SW1 and SW3 are off. This adds R4 in between the divider junction and V_{ref}. Because R4 is a higher resistance than R5, less V_{ref} bias voltage is mixed with V_{in}. This gives greater resolution down to -20V. Going below -20V will make V_{meas} go to -3mV, and the microcontroller software will display a bottom-most value just below -20V. If this happens, auto-ranging should switch such a condition to the ±60V DC range. (See Chapter 7.)

60V AC Range: The shield circuit is in the same state as in the ±60V DC Range. Different calculations are used to determine the AC voltage.

20V AC Range: The shield circuit is in the same state as in the ±20V DC Range. Different calculations are used to determine the AC voltage.

CHAPTER 2: AUXILIARY HARDWARE CONSIDERATIONS

Header pins were soldered to all of the necessary pads of the LM3S8962 board. The shield circuit was constructed for one channel on a Twin Industries TW-E41-102B breadboard, and later expanded to four channels using a Twin Industries TW-E41-1060 breadboard. Once prototyping was completed, the shield was soldered to a Twin Industries 8300SB1 protoboard. The protoboard, LM3S8962 board, and associated connectors were installed inside a small plastic enclosure, and attached with a DB25 connector. Where necessary, the smaller "102B" breadboard was utilized externally as a voltage divider for the four input voltages when testing, since only one DC power supply was available.

The enclosure has five banana jacks. Four of these are for connecting four voltage inputs to the shield circuit, and one is a common ground. Ground is tied to the ground of the ADC, which is also connected internally in the LM3S8962 board to each other ground.[3 p.24] Therefore, it's important that ground of the items being measured be the same, and that the ground does not have any live voltage present. Some commercial digital voltmeters have separate reference grounds and electronic grounds; [7 p.141] this reveals one drawback to using this particular microcontroller as a voltmeter.

Supplied test power came from an HP 6299A DC Power Supply. Supplied AC test power came from a variAC. A calibrated Fluke 8060A digital multimeter was used as a control measurement and baseline to assess the accuracy of the resulting project.

CHAPTER 3: DESCRIPTION OF USER INTERFACE

DISPLAY

The OLED display of the evaluation board is laid out according to Figure 2. Figure 2 is annotated and labeled with red arrows and letters, and below Figure 2 are descriptions of each label.

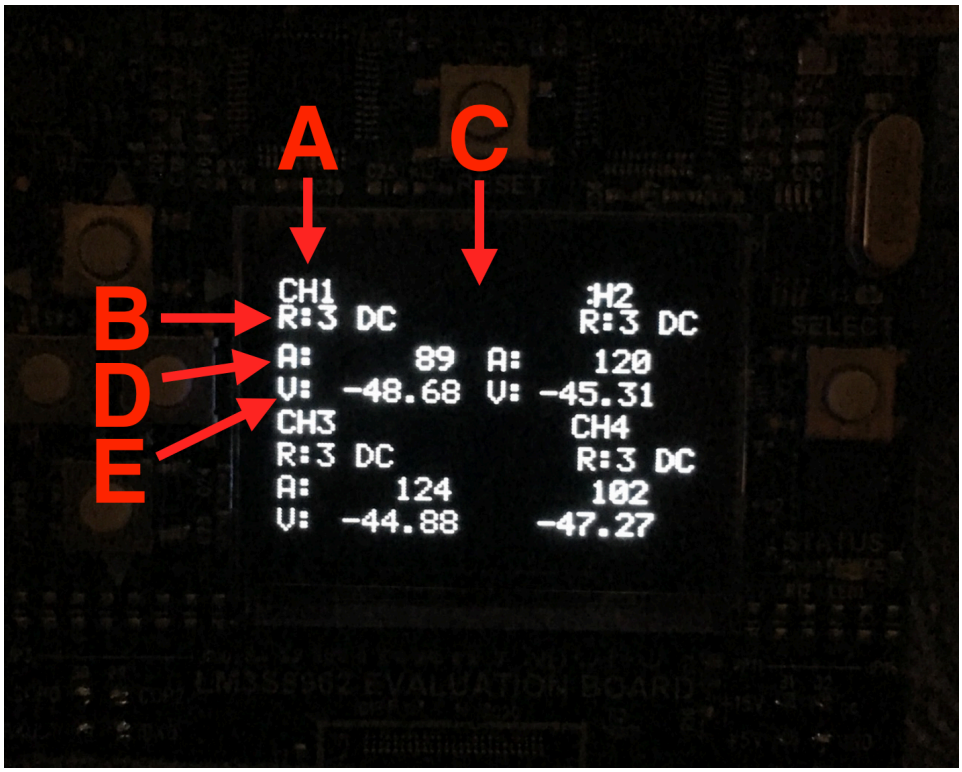


FIGURE 2: Display Elements

A: The display accommodates 4 channels, one per quadrant. This is a static label showing which channel is shown in that quadrant.

B: The range of the channel is displayed here. The range display has the following key, shown in Table 2:

Display Text	Meaning
`R: 1 DC`	+60V DC range
`R: 2 DC`	+20V DC range
`R: 3 DC`	±60V DC range
`R: 4 DC`	±20V DC range
`R: 5 AC`	60V AC range
`R: 6 AC`	20V AC range

TABLE 2: Range Display Description

C: When too much voltage is detected on any channel, "`OVERLOAD!!`" text appears here

D: This is the numerical value output by the ADC. It is there primarily for demonstration and debugging purposes.

E: Calculated voltage measurement is shown here.

BUTTONS

The LM3S8962 has four buttons (up/down/left/right) assigned to GPIO Port E pins 0, 1, 2, and 3 respectively. Channel 1 is assigned to the "Up" button, Channel 2 is assigned to the "Left" button, Channel 3 is assigned to the "Down" button, and Channel 4 is assigned to the "Right" button. When an appropriate button is pressed, the auto-range mode is toggled between DC and AC for the respective channel. For more on auto-ranging, see Chapter 7.

The "Reset" button resets the entire board, which also resets the range to +60V.

CHAPTER 4: STARTUP SEQUENCE AND EXECUTION

ENABLING PERIPHERALS

All peripherals are enabled, including the RIT128x96x4 display, GPIO B, D, E, F, G, and the ADC.[Appendix C lines 1881-1899]

The shield circuit is set to the +60V range for all channels via configuring GPIO pins to turn on/off the correct MOSFET switches for this range. [Appendix C lines 1916-1926]

The ADC is configured for a default rate of 125k samples per second, though the while() loop only captures a sample once every 40ms. The ADC samples on sequence 2, which has 4 steps[5 p.399], each assigned to channels 1, 2, 3, and 4 respectively. [Appendix C lines 1933-1945] Hardware oversampling is enabled with an oversampling factor of 6[5 p.400]. The ADC interrupt is cleared, and initial labels are drawn on the display. [Appendix C lines 1955-1967]

Ranges are stored in global integer variables, and initialized at "`1`" to equal range 1 (+60V DC). All counters are initialized at "`0`". [Appendix C lines 27-153]

THE MAIN WHILE() LOOP

The main `while()` loop first triggers a sample, which produces an ADC interrupt. Each sample is stored in the `ulADC0_Value[]` array. [Appendix C lines 1969-1978] Position 0 of the array is the ADC value from channel 1, position 1 is channel 2, and so on. The value from the ADC is a 10-bit number of type `long`[5 p.399]. The 4 sampled

The remainder of this section will describe what happens with channel 1. It should be understood that the same process is repeated for each of the 4 channels.

AVERAGING

The `DCAvg1()` method stores the sample in an array called `ADCValueAvg1[]` which holds up to 80 samples. [Appendix C line 947] `DCAvg1()` is equivalent to a FIFO (first-in first-out) buffer in how it loads and replaces its values in a 80-sample cyclic fashion. The `DCAvg1()` method then calculates the sum of all 80 samples, and returns the sum. [Appendix C line 961] The `while()` loop then takes the returned sum, and divides it by 80, storing this value in `ch1ADCDisplay`. [Appendix C lines 2069-2073]

The `DCAvg1()` method was created to address noise and instability in the observed measurement on the display. It takes a few seconds for the ADCValueAvg1[] array to fill up, and so the drawback of using this method is that the displayed voltage takes a few seconds to read accurately. The benefit of this method is that it greatly stabilizes the voltage reading so that little to no "jitter" in the value is observed for steady unchanging DC voltages.

MAXIMUM AND MINIMUM CALCULATION

By using an 80-sample long buffer to store readings, this also presents an opportunity to find the minimum and maximum ADC readings in that 2 to 3 second period of time, storing these in the `maximum1` and `minimum1` variables. A `for()` loop is used to cycle through all the values in `ADCValueAvg1[]` to find the minimum and

maximum sample. These values are useful for both auto-ranging, and for AC measurement. [Appendix C lines 2075-2088]

CHECK FOR BUTTON PRESSES

The `while()` loop then runs `if()` statements to determine whether or not a button has been pressed. If so, the appropriate channel is toggled between DC and AC auto-ranging. [Appendix C lines 1980-2067]

AUTO-RANGING

Auto-ranging is accomplished through a large number of small methods. For a given range (1 through 6), there is a dedicated method that determines if and how the range should be changed. In total, there are 24 different methods responsible for auto-ranging, but at any given point in time, only one is running per channel. [Appendix C lines 2130-2160, and 1021-1232] For more on Auto-ranging, see Chapter 7.

UPDATE DISPLAY

Depending on whether the channel is in DC or AC measurement, `if()` statements determine the correct ADC and voltage reading to display. When in DC mode, the "`A:`" field displays the `ch1ADCDisplay` value, which is an average DC calculation. [Appendix C line 2268] The `ch1ADCDisplay` and current range and channel are then passed to the `adj_n()` method, [Appendix C line 2269] which contains a series of functions that translate the ADC reading to real-world voltages through mathematical

equations. Some of these functions are linear, some quadratic, and some cubic. [Appendix C lines 156-316] The result returned is displayed in the "`V:`" field. [Appendix C line 2269] When in AC mode, the "`A:`" field displays the difference between `maximum1` and `minimum1`. [Appendix C line 2262] This difference is passed to the `adj_n()` method[Appendix C line 2261], which has its own separate calculations for translating these values to real-world AC voltages, then shown on the "`V:`" field.

Finally, the ADC interrupt is cleared in case it has not already cleared after finishing a sample. [Appendix C line 2310]

TIMING DELAYS

The end of the `while()` loop has a delay. [Appendix C lines 2313-2317] Changing the value of the delay speeds up or slows down the speed at which the ADC triggers. This is currently set to `SysCtlDelay(SysCtlClockGet()/75)`, which delays for 40ms. If the display refreshed once every 40ms, this would be much too fast to make the display readable, so a counter is written in the display's `if()` statement which only refreshes the display once every 7 sample counts (280ms). [Appendix C line 2253]

CHAPTER 5: TRANSLATION TO REAL-WORLD VOLTAGES

NON-LINEARITY IN THE SHIELD CIRCUIT

The ADC values for each range had to be translated into actual measured voltages. In a voltage-dividing network consisting only of resistors, this would only require one linear function. However, the shield circuit uses a zener diode to serve as a voltage clamp. As V_{in} increases, V_{meas} approaches approximately 2.7V, much like a horizontal asymptote. To illustrate this effect, observe the graph in Figure 3:

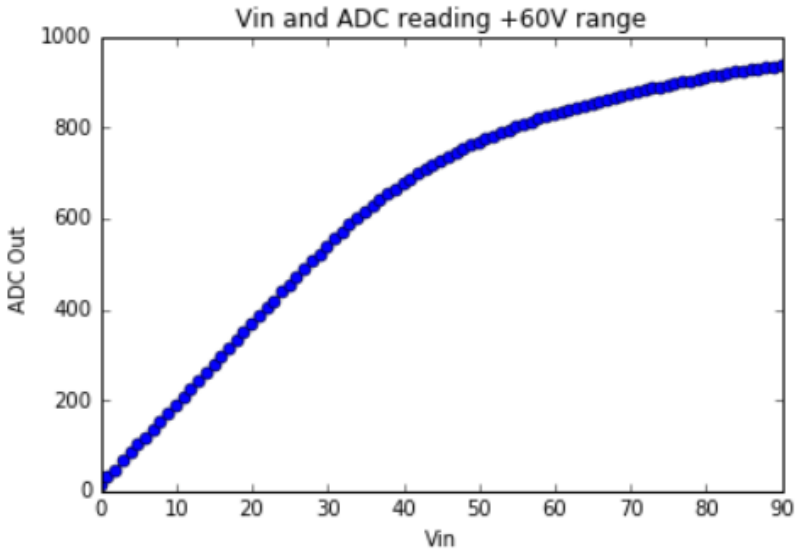


FIGURE 3 - V_{in} and ADC Reading +60V

The X-axis is V_{in}, and the Y-axis is ADC measurement. The maximum ADC measurement is 1023, which equates to $3V_{meas}$. Regardless of which zener diode was chosen, the effect was the same. To have a higher V_{meas} but not get too close to the

maximum of $3V_{meas}$ that the evaluation board could handle, a 3.3V zener diode was chosen which allowed V_{meas} to max out at approximately 3V.

The ADC value is a 10-bit floating-point number[5 p.399] ranging between zero and 1023. To begin assessing the non-linearity, readings were taken at 1V increments from 0 to 60 volts, with the +60V range selected. The resulting graph, when mapped with ADC in the X axis and V_{in} on the Y axis, looks like Figure 4.

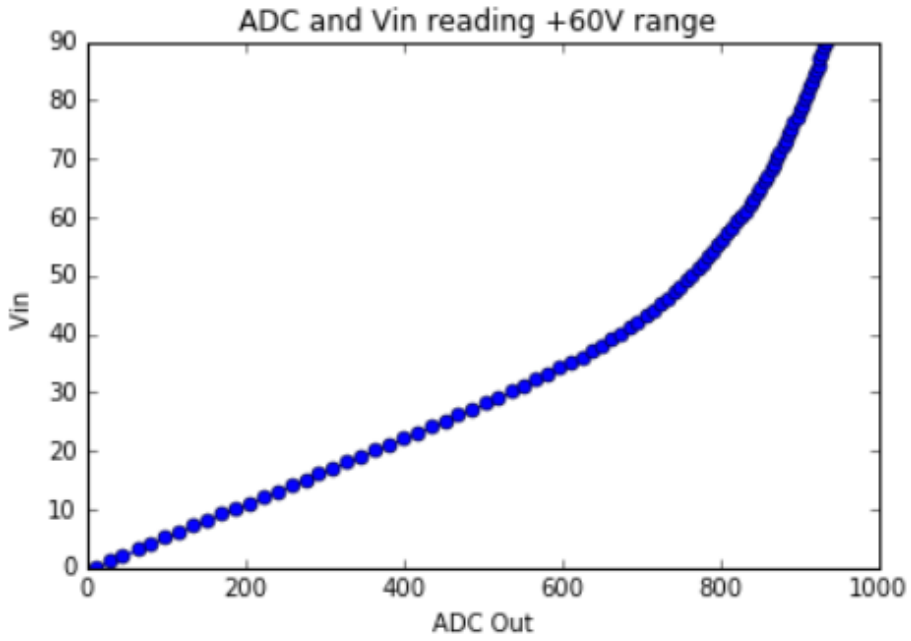


FIGURE 4 - ADC and V_{in} Reading +60V

The first section of this graph appears linear. At a certain point (after about X = 500), the voltage present at the cathode of the zener diode reaches 1V, and the zener diode reaches its forward voltage, beginning to conduct. Therefore, the graph becomes non-linear. The best fit was found with a cubic function after this threshold. A similar graph structure was observed for the 20V range, though with different cutoff points at

which the forward voltage occurs, and thus, different transition points from linear to cubic.

ALTERNATIVES FOR LINEARITY

Removing the zener diode from the circuit and using higher value resistors could have been one solution that produced linearity, but this would run the risk of damaging the ADC input of the board when too high voltage is fed to the shield, or if an incorrect range was selected. This would have also made auto-ranging dangerous since the shield circuit might be in a suboptimal range for some period of time while the auto-range algorithm runs.

An alternative approach would have been to use opamps to amplify a very low voltage from a conservative voltage divider as previously mentioned. A properly selected opamp may be capable of taking a very low voltage from said voltage network and amplifying it to an appropriate voltage for the ADC.

ADAPTING FOR NON-LINEARITY IN SOFTWARE

The `adj_n()` function has separate functions for each range. For every range, separate functions translate the various segments of the graph, depending on the determined thresholds of each graph based on the ADC reading.

Manual readings of V_{in} were taken with a Fluke 8060A meter, and plotted against the resulting ADC values on the display. For the +60V range, 1V increments were taken across the full range, up to 90V. For the 20V range, 0.5V increments were taken. The

overall graph was plotted and observed in iPython Notebook. Here, the linear/cubic cutoff point was visually determined. (The cutoff point was always slightly above the $1V_{meas}$ forward-conducting point.) Then, the measured plots for each portion of the graph were input with Numpy.polyfit() to determine the coefficients for each function; one was done for the linear regression, and another done for the cubic regression. For full details on how these calculations were made, refer to the iPython Notebooks shown in Appendix B.

Figures 5 and 6 show each of the two segments of the +60V range. The red dots are the actual ADC readings taken, and the blue line is the resulting regression function superimposed on the plot. This visually demonstrates the goodness of fit.

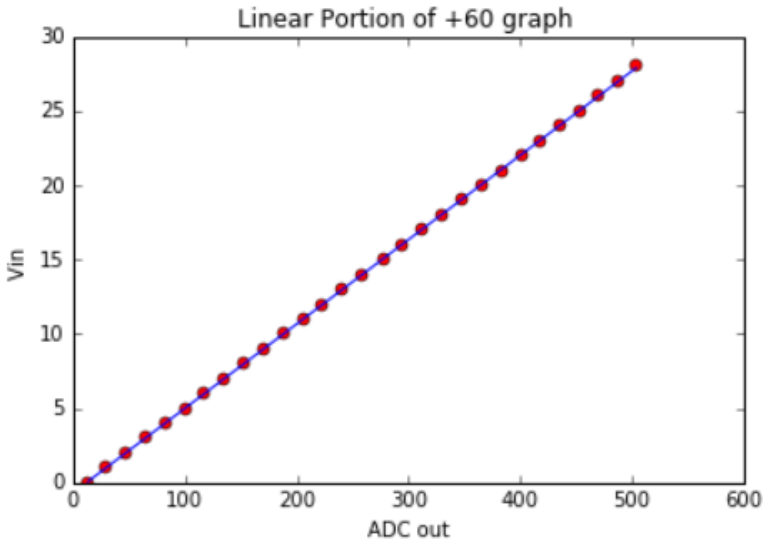


FIGURE 5 – Linear portion of +60 graph

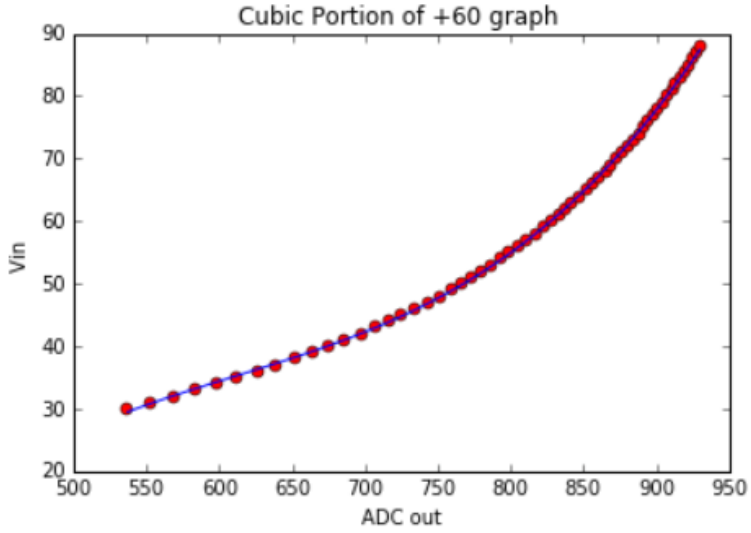


FIGURE 6 - Cubic Portion of +60V Graph

Interesting behavior occurs when switching to the ±60V and ±20V ranges. The plot of ADC values versus V_{in} for the ±60V range is shown in Figure 7, and in Figure 8 for the ±20V range.

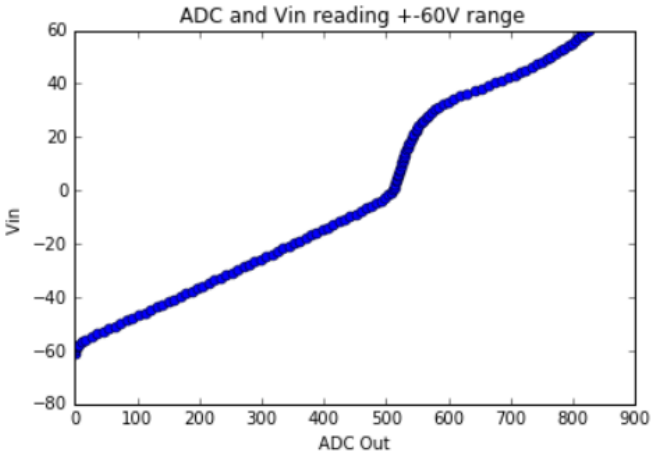


FIGURE 7 - ADC and Vin Reading ±60V Range, DC Input

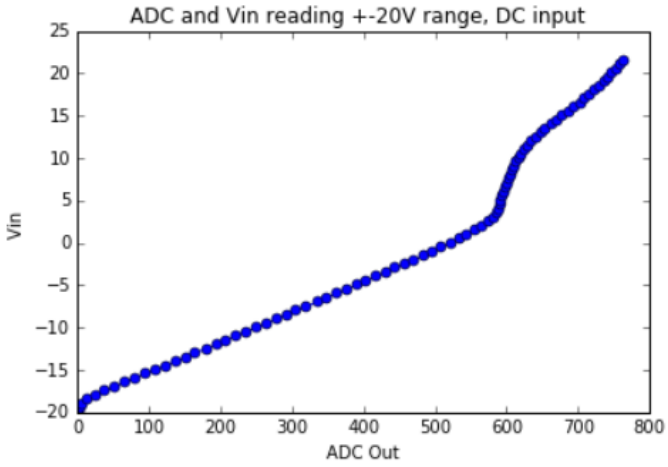


FIGURE 8 - ADC and Vin Reading ±20V Range, DC Input

These unusual curves were overcome by using 4 separate functions, dividing the data into 4 segments. They vary between linear, quadratic, and cubic functions depending on which had the best fit in Python.

Even with carefully matched components for the shield circuit on channels 2, 3, and 4, minor differences (especially between the zener diodes) caused different behavior between different channels. Recycling the same `adj_n()` equations for all 4 channels proved inaccurate. So, each channel has its own functions for calculating real-world voltage, and has this for each range. Voltage sweeps were done (every 5 volts in the +60V and ±60V range, and every 2 volts in the +20 and ±20V ranges), the ADC output of the sweeps recorded, and separate regressions made for each channel and each range. Commercial meter manufacturers likely have computerized automation of this calibration curve.

CHAPTER 6: CALCULATING AC VOLTAGE

AC voltage is measured by using already-existing code components that smoothed out the DC reading. An 80-length FIFO buffer (`ADCValueAvg1[]`) was made for smoothing out the DC readout. [Appendix C lines 947-962] Since the readout spanned the last few seconds, some readings in the buffer would be higher, and lower than the average. This fluctuation is characteristic of AC voltage, and can be measured by calculate the difference between the maximum and minimum reading for a period of time. The `maximum1` and `minimum1` values were already calculated in their respective `for()` loops every cycle count. [Appendix C lines 2075-2088]

When the code for refreshing the display recognizes that a channel is in AC mode, the difference between `maximum1` and `minimum1` is shown on the display, and also sent to `adj_n()` to the translate to real-world voltage. [Appendix C lines 2259-2264]

A variAC was used as an AC signal source. One leg of AC was V_{in} input, and earth ground was ground. Although a relatively noisy source of AC voltage, it did stabilize after some time, and provide a meaningful readout. Given that there was no bias on V_{in}, the AC input is centered around 0V. Therefore it is likely that some error will occur when measuring AC voltages that are superimposed on a DC signal source. Furthermore, the upper half of the AC wave has less resolution than the lower half. Superimposed DC on an AC input will have different variations between its maximum and minimum readings. This is a known limitation of the current design.

Figures 9 and 10 plot the ADC readings versus V_{in} (AC) for 60V and 20V ranges respectively. As before, a segmented regression function had to be calculated for a reasonable fit with the data points.

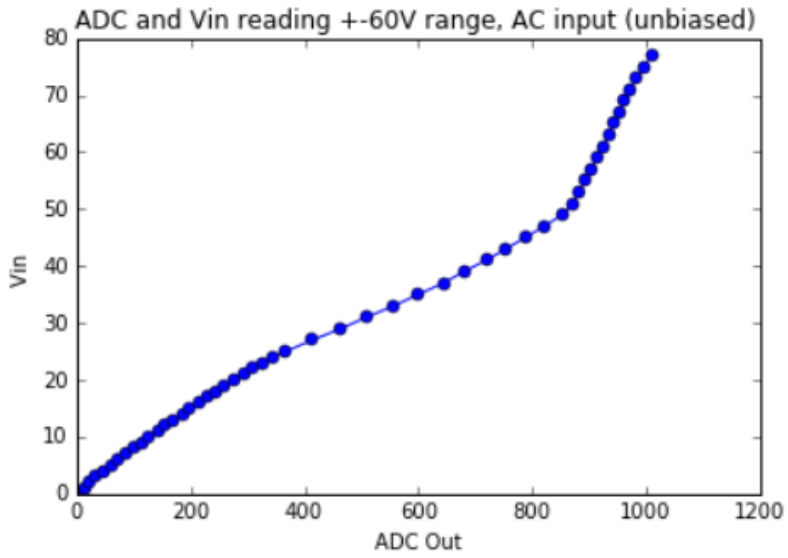


FIGURE 9 - ADC and Vin Reading ±60V Range, AC Input (unbiased)

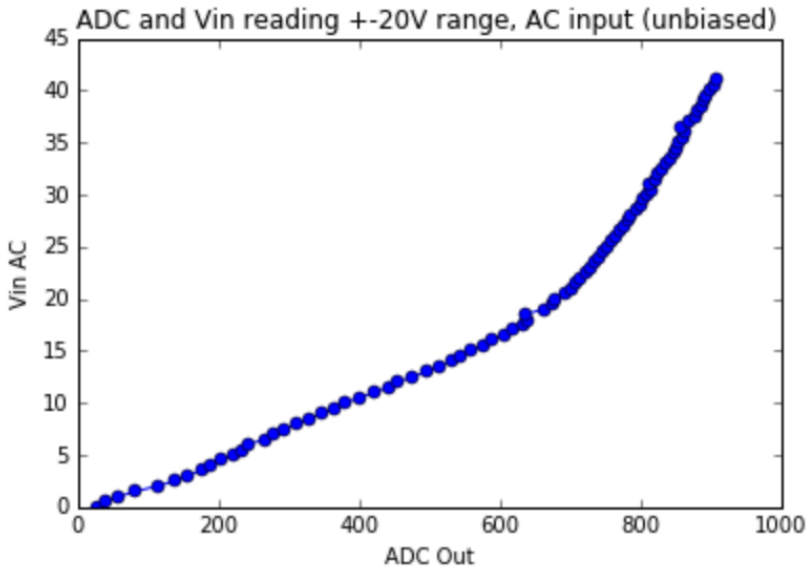


FIGURE 10 - ADC and Vin Reading ±20V Range, AC Input (unbiased)

CHAPTER 7: AUTO-RANGING

Auto-ranging automatically puts the meter into the range that most closely matches the voltage input without the user having to do so manually, or know what the voltage should be. Implementing auto-ranging was especially important with 4 channels, and 6 total ranges per channel. Now that reliable DC and AC readouts were shown, it was possible to write an algorithm that, for a given range, could decide whether or not it would be wise to change to a different range, and if so, automatically decide which range to change to.

Table 3 shows the Auto-Range algorithm in the most basic of terms:

CURRENT RANGE	CONDITION	ACTION
1 (+60 DC)	If ADCValue < 0V threshold	Switch to Range 3 (±60V)
	If ADCValue < 20V threshold	Switch to Range 2 (+20V)
2 (+20 DC)	If ADCValue < 0V threshold	Switch to Range 3 (±60V)
	If ADCValue > 20V threshold	Switch to Range 1 (+60V)
3 (±60 DC)	If ADCValue > 0V threshold	Switch to Range 1 (+60V)
	If (min > -20V) & (max < 20V)	Switch to Range 4 (±20V)
4 (±20 DC)	If ADCValue > 0V threshold	Switch to Range 1 (+60V)
	If ADCValue < -20V treshold	Switch to Range 3 (±60V)
5 (±60 AC)	If max-min < 20VAC	Switch to Range 6 (±20V AC)
6 (±20 AC)	If max-min > 20VAC	Switch to Range 5 (±60V AC)

TABLE 3: Auto-Range Algorithm

It takes some time for the voltage readout to stabilize once a range is selected. In some cases, this stabilization can take several seconds as both the electronics of the shield circuit stabilize, and the variables of the code change their mathematical calculations. Therefore it was desirable for the auto-range algorithm to give some time for this stability to take place, and to only consider switching ranges after this amount of time had passed. To that end, every channel has 6 dedicated methods (one method for each range), which will run only when a particular range is actively selected. Each of these methods has 2 or 3 counters. There is a master counter that will increment on each cycle count[Appendix C line 1024], and separate condition counters for each switching condition (2 conditions for each DC range, and 1 condition for each AC range.) [Appendix C lines 1027-1028] All 3 of these counters reset to zero once the master counter reaches 240. In other words, the entire switching method will reset to zero approximately every 5 seconds.

If the circumstances of the current reading match criteria for that of the switching condition, the switching condition counter is incremented. [Appendix C lines 1030-1052] If the condition counter reaches a count of 120, then the range value is switched, and the appropriate GPIO pins for MOSFET switching of the shield circuit are switched on or off. [Appendix C lines 1035-1040, and 1048-1051] The next cycle of the while loop runs the auto-range method for the current range and channel, and the process begins again. The goal of all of these counters and delays is that a channel should remain in the most appropriate measurement range indefinitely, for any given steady voltage input.

The buttons are used to toggle between DC and AC measurement. If currently in any of the 4 DC ranges, pressing the appropriate button changes to Range 5 for that

channel (60V AC). [Appendix C lines 1980-1990] Then, if the auto-ranging algorithm decides the AC voltage is lower than 20, it will change to Range 6 (20V AC). [Appendix C lines 1180-1205]

Pressing the same button again toggles back to DC, Range 1 (+60V DC). [Appendix C lines 1992-1998] The auto-ranging algorithm resumes with the auto-range method for Range 1, changing to the appropriate DC ranges when needed.

CHAPTER 8: ADDITIONAL NOTES OF THE EXECUTION CODE

The code for the project is written in C, and done so via Keil (an ARM MDK) version 4.74. This project uses the following GPIO ports:

GPIO B, D, F, G are used for the switching MOSFETS to change the range.

GPIO E is used for the up/down/left/right buttons for the user to toggle DC/AC auto-ranging for each channel. See Appendix A for a full list of GPIO pins and DB25 connector pin assignments.

One challenge was correctly displaying an ever-changing "long" ADC numerical reading onto the RIT 128x96x4 display. The default driver for this display provided by Stellarisware only accommodated printing constant strings. Though there are many ways to translate `long` to `string`, a different version of the RIT driver meant for the LM3S1968 board was found from Dr. Jonathan Valvano's posted rit128x96x4.c code[8]. Rather than use this in its entirely, only the `UInt2Str4()` function was placed into the code, which converts an unsigned integer (0 to 9999) to a displayed string. This allows ADC values to output to the display, though they are not translated to real voltages.[Appendix C lines 846-909]

Next, translation functions were written in the `adj_n()` method. The ADC value and current range is fed into this method as an argument. For each range, the ADC value is compared to an overload threshold, and "overload" is displayed if appropriate. If there is no overload, a set of `If()` statements decides whether to perform a quadratic, linear, or cubic translation. Depending on the pre-decided (hard-coded) thresholds, the

appropriate translation function is done, to thus create and return "`n`", the actual voltage string displayed.

"`n`" should be precise and contain two decimal places, so a different method for displaying "`n`" is used, also derived from Dr. Jonathan Valvano's posted rit128x96x4.c code[8]. The `Fix2Str()` method takes a fixed-point number and displays it.[Appendix C lines 747-809] For example, "`12345`" is displayed as "`123.45`". This is why each translation function was multiplied by 100 so that they are displayed with the correct decimal place.

CHAPTER 9: ACCURACY ASSESSMENT

Upon satisfactory completion of the project, an accuracy assessment was done which sent arbitrarily selected voltages between -60 and +60 to each channel, comparing how that voltage read on a Fluke 8060A meter, versus how the voltage displayed on each LM3S8962 channel. The absolute value of the difference in voltage was calculated for each measurement, and for each channel. Average error was calculated per-channel.

Table 4 shows an overall average DC error of 0.20V. The highest error was observed in Channel 4 when testing -1.63 VDC, and 60.86 VDC.

Fluke Measurement	Ch 1 Readout	Error (volts)	Ch 2 Readout	Error (volts)	Ch 3 Readout	Error (volts)	Ch 4 Readout	Error (volts)
-56.12	-55.74	**0.38**	-56.51	**0.39**	-56.5	**0.38**	-55.89	**0.23**
-34.21	-33.69	**0.52**	-34.22	**0.01**	-34.49	**0.28**	-34.21	**0.00**
-19.35	-19.35	**0.00**	-19.43	**0.08**	-19.55	**0.20**	-19.28	**0.07**
-7.37	-7.18	**0.19**	-7.3	**0.07**	-7.4	**0.03**	-7.3	**0.07**
-1.63	-1.64	**0.01**	-1.76	**0.13**	-1.9	**0.27**	-0.42	**1.21**
0.01	0.09	**0.08**	0.05	**0.04**	0.03	**0.02**	0.15	**0.14**
0	0.03	**0.03**	0	**0.00**	0.03	**0.03**	-0.03	**-0.03**
3.34	3.37	**0.03**	3.31	**0.03**	3.3	**0.04**	3.27	**-0.07**
15.09	15.10	**0.01**	14.91	**0.18**	14.97	**0.12**	15.03	**-0.06**
25.78	25.91	**0.13**	25.8	**0.02**	25.43	**0.35**	26.06	**0.28**
35.56	36.07	**0.51**	35.41	**0.15**	35.51	**0.05**	35.91	**0.35**
45.46	45.43	**0.03**	45.59	**0.13**	45.16	**0.30**	45.63	**0.17**
55.97	55.45	**0.52**	56.46	**0.49**	56.18	**0.21**	56.03	**0.06**
60.86	60.69	**0.17**	61.54	**0.68**	61.15	**0.29**	62	**1.14**
Overall AVG DC Error: 0.20 V	**Ch 1 DC AVG Error**	**0.19**	**Ch 2 AVG Error**	**0.17**	**Ch 3 AVG Error**	**0.18**	**Ch 4 AVG Error**	**0.25**

TABLE 4: DC Accuracy Assessment
Fluke Measurement made on Vin, Readouts were on OLED

Another assessment was done for AC measurement. See Table 5. Four arbitrarily selected source AC voltages were fed to each of the 4 channels independently. The results show an overall average error of 1.90VAC. This larger error is partly due to the noisy source of AC voltage, and partly due to how the software reacts to such noise. A noisy signal may produce spurious peaks or troughs in the 80-sample period of time where the `DCAvg` buffers are filled, which may cause erroneous minimum or maximum readouts used to calculate the AC voltage. A better AC signal source can alleviate this problem, combined with studying a method of ADC measurement that can ignore or filter out spurious peaks or troughs.

Fluke Measurement	Ch 1 Readout	Error (volts)	Ch 2 Readout	Error (volts)	Ch 3 Readout	Error (volts)	Ch 4 Readout	Error (volts)
12.5	12.21	**0.29**	12.29	**0.21**	12.28	**0.22**	12.04	**0.46**
29.08	27.64	**1.44**	29.01	**0.07**	28.71	**0.37**	30.74	**1.66**
45.68	45.16	**0.52**	43.53	**2.15**	42.26	**3.42**	43.48	**2.20**
60.04	57.13	**2.91**	54.16	**5.88**	53.51	**6.53**	57.98	**2.06**
Overall AVG AC Error: 1.90 V	**Ch 1 AC Avg Error**	**1.29**	**Ch 2 AVG Error**	**2.08**	**Ch 3 AC Avg Error**	**2.64**	**Ch 4 AC Avg Error**	**1.60**

TABLE 5: AC Accuracy Assessment
Fluke Measurement made on Vin, Readouts were on OLED

CONCLUSION

The Cortex M3 on the LM3S8962 microcontroller has a 4 channel ADC that measures 10 bits of resolution between 0 and 3 Volts. The high resolution of measurement present in this ADC is why the ADC was effective as a voltmeter.

A shield circuit was constructed with commodity electronic components to bias the input voltage where needed, and to protect the ADC from damage. The shield circuit can withstand positive or negative voltages between -60V and +100V, though accuracy is only assured within ±60V. The shield, connectors, and corresponding wires were constructed inside an enclosure to protect the user from accidental electrocution.

The code for the LM3S8962 was written in C using the Keil MDK. The report explains in detail the execution of the code including enabling of peripherals, initialization of all counters and shield circuit state, and periodic measurements taken from the ADC. The data gathered from each ADC channel is put through a number of functions to translate to real-world voltage.

The RIT128x96x4 OLED display shows the necessary information from all 4 channels including the channel number, the range, ADC readout, translated voltage readout, as well as overload warnings.

Voltage translation functions compensated for most hardware non-linearity, and help model the project's behavior to that of a calibrated Fluke 8060a digital multimeter. Python code was used to find translation functions with the best fit, and to decide the correct ADC thresholds for each translation function. Over-fitting was avoided by restricting the polynomial functions to fitting a linear, quadratic, or cubic regression.

A secondary FIFO took the average ADC readout over a period of a few seconds in order to reduce jitter on the displayed voltage reading. Taking the difference between the maximum and minimum of this buffer's values produces a measurement for AC voltage, which was translated to real-world AC voltage.

Auto-ranging was achieved through an algorithm that decides when to switch out of a given range, and which range to switch to.

Accuracy was assessed at an average of ±0.2V for DC measurements, and within ±1.9V for AC measurements. Better accuracy may be accomplished with less noisy voltage sources, and an alternative shield circuit design with more consistent component behaviors between channels.

This project may provide a newfound appreciation for the research and development that goes into today's commercial multimeters. The multitude of issues that were discovered and compensated for during the design of this project show that commercial voltmeters may be well worth the expense for those who wish to have an off-the-shelf solution that is guaranteed to be accurate.

Still, the features of this project together produce a surprisingly accurate 4 channel auto-ranging digital voltmeter, which may be significantly less expensive than a commercial product with similar features.

Appendix A – Shield Connector Pin Assignments

GPIO Pin	LM3S8962 Breakout Pin Label	DB25 Connector Pin	Shield Connection
G0	PG0	19	CH1 SW1
D2	U1RX	5	CH1 SW2
D3	U1TX	18	CH1 SW3
D4	CCP0	14	CH1 SW1
F2	LED1	6	CH2 SW2
D7	IDX0	1	CH2 SW3
F1	IDX1	25	CH3 SW1
F3	LED0	10	CH3 SW2
B1	PWM3	24	CH3 SW3
B4	C0-	22	CH4 SW1
B5	C1-	8	CH4 SW2
B6	C0+	23	CH 3 SW3
-	ADC0	2	CH 1 Vmeas
-	ADC1	16	CH 2 Vmeas
-	ADC2	3	CH 3 Vmeas
-	ADC3	17	CH 4 Vmeas
-	GND*	4, 7, 9, 11, 12, 13, 15, 21	GND*
E0	up button	-	-
E1	down button	-	-
E2	left button	-	-
E3	right button	-	-
-	3.3V	20	Vref

All grounds internally connected within LM3S8962 board

Appendix B – iPython Notebooks to Calculate Real-World Voltage Translation Functions

In [1]:
```
import matplotlib.pyplot as plt
import csv
import numpy as np
from numpy import arange,array,ones,linalg
from pylab import plot,show
from scipy.optimize import curve_fit
%matplotlib inline
```

In [2]:
```
# datax=np.genfromtxt('60vdata.csv', delimiter=",")
datax=np.genfromtxt('60vdata.csv', delimiter=",", usecols=(0))
datay=np.genfromtxt('60vdata.csv', delimiter=",", usecols=(1))
```

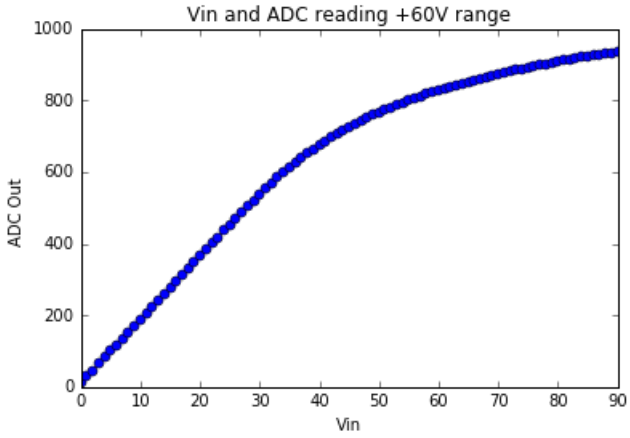
In [4]:
```
print datax
```

```
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.  12.  1
3.  14.
  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  2
8.  29.
  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  4
3.  44.
  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56.  57.  5
8.  59.
  60.  61.  62.  63.  64.  65.  66.  67.  68.  69.  70.  71.  72.  7
3.  74.
  75.  76.  77.  78.  79.  80.  81.  82.  83.  84.  85.  86.  87.  8
8.  89.
  90.]
```

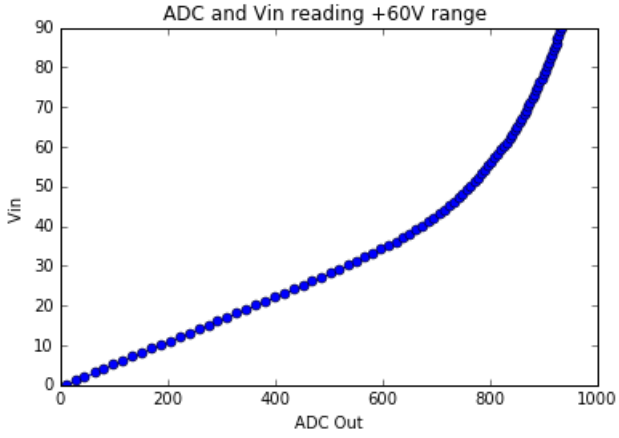
In [5]:
```
print datay
```

```
[  13.   29.   46.   65.   82.  100.  117.  135.  153.  170.  188.
206.
  223.  241.  259.  277.  294.  312.  330.  348.  365.  383.  401.
418.
  436.  453.  470.  487.  504.  521.  537.  553.  568.  584.  598.
612.
  626.  639.  652.  664.  675.  686.  697.  707.  716.  725.  734.
743.
  751.  759.  766.  773.  780.  787.  793.  799.  805.  811.  817.
822.
  828.  833.  838.  842.  847.  852.  856.  860.  865.  869.  873.
876.
  880.  884.  888.  891.  894.  898.  901.  904.  907.  911.  913.
917.
  920.  922.  925.  927.  930.  933.  936.]
```

In [7]:
```
plt.title ('Vin and ADC reading +60V range')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(datax, datay, 'o-')

plt.show()
```



In [8]:
```
plt.title ('ADC and Vin reading +60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(datay, datax, 'o-')

plt.show()
```



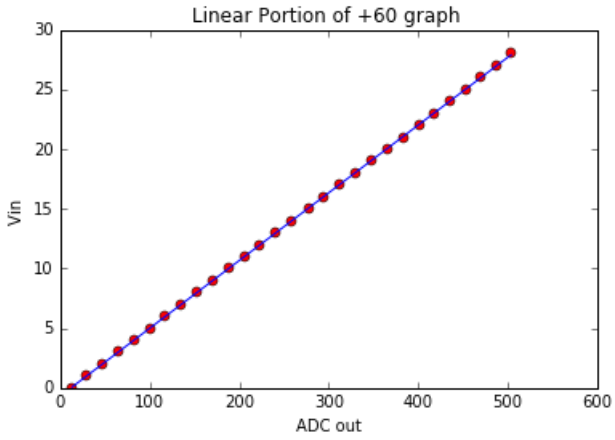
In [9]:
```
z1 = np.polyfit(datay[0:29],datax[0:29],1)

print z1
```

```
[ 0.05667105 -0.66607681]
```

In [10]:
```
plin = np.poly1d(z1)
```

In [11]:
```
plt.title ('Linear Portion of +60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(datay[0:29], datax[0:29],'ro')
plt.plot(datay[0:29], plin(datay[0:29]), '-')
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x10bcd3610>]
```

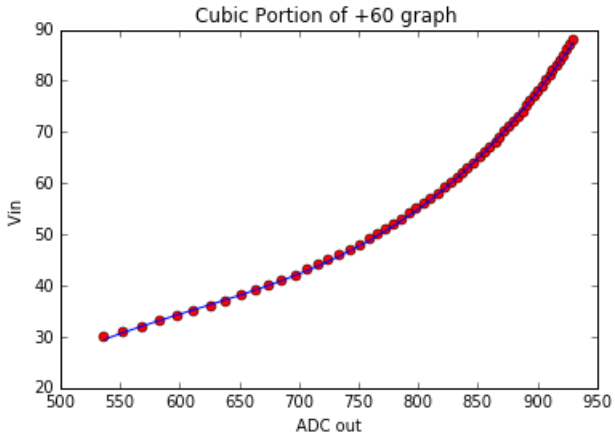


In [12]:
```
z2 = np.polyfit(datay[30:89],datax[30:89],3)

print z2
```

```
[  8.98034069e-07  -1.64569796e-03   1.07759340e+00  -2.13673430e+02
]
```

In [13]:
```
pcub = np.poly1d(z2)
```

In [14]:
```
plt.title ('Cubic Portion of +60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(datay[30:89], datax[30:89],'ro')
plt.plot(datay[30:89], pcub(datay[30:89]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```



In [15]:
```
data20x=np.genfromtxt('20vdata.csv', delimiter=",", usecols=(0))
data20y=np.genfromtxt('20vdata.csv', delimiter=",", usecols=(1))
```

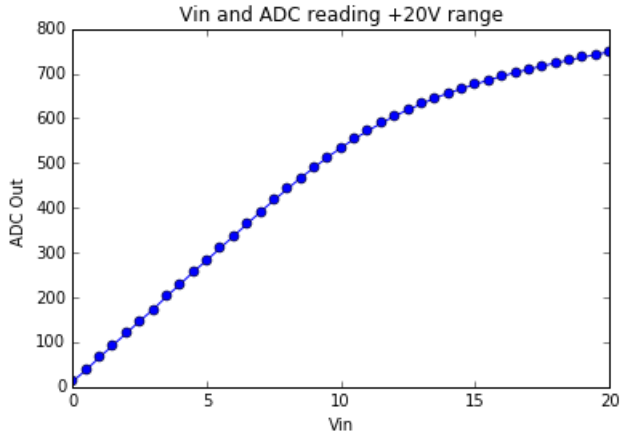
In [16]:
```
print data20x
```

```
[  0.    0.5   1.    1.5   2.    2.5   3.    3.5   4.    4.5   5.
5.5
   6.    6.5   7.    7.5   8.    8.5   9.    9.5  10.   10.5  11.
11.5
  12.   12.5  13.   13.5  14.   14.5  15.   15.5  16.   16.5  17.
17.5
  18.   18.5  19.   19.5  20. ]
```

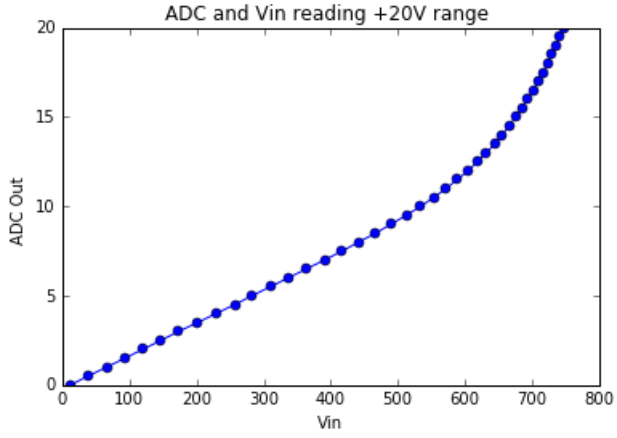
In [17]:
```
print data20y
```

```
[  12.   38.   66.   93.  120.  147.  173.  202.  229.  257.  283.
310.
  337.  364.  391.  417.  442.  467.  490.  513.  534.  554.  572.
589.
  605.  619.  632.  645.  656.  666.  676.  685.  694.  702.  709.
717.
  723.  730.  736.  742.  748.]
```

In [18]:
```
plt.title ('Vin and ADC reading +20V range')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(data20x, data20y, 'o-')

plt.show()
```



In [19]:
```
plt.title ('ADC and Vin reading +20V range ')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(data20y, data20x, 'o-')

plt.show()
```



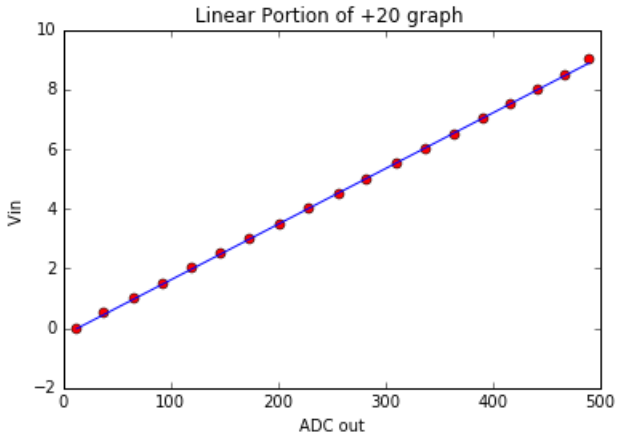
In [20]:
```
z3 = np.polyfit(data20y[0:19],data20x[0:19],1)

print z3
```

```
[ 0.01862997 -0.24377945]
```

In [21]:
```
plin20 = np.poly1d(z3)
```

In [22]:
```
plt.title ('Linear Portion of +20 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20y[0:19], data20x[0:19],'ro')
plt.plot(data20y[0:19], plin20(data20y[0:19]), '-')
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x10c094610>]
```

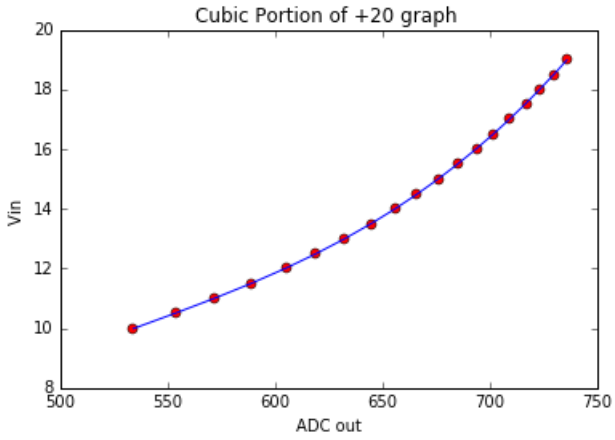


In [23]:
```
z4 = np.polyfit(data20y[20:39],data20x[20:39],3)

print z4
```

```
[  4.47800473e-07  -7.16941086e-04   4.08798967e-01  -7.20598349e+01
]
```

In [24]:
```
pcub20 = np.poly1d(z4)
```

In [25]:
```
plt.title ('Cubic Portion of +20 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20y[20:39], data20x[20:39],'ro')
plt.plot(data20y[20:39], pcub20(data20y[20:39]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```



In [26]:
```
data60bx=np.genfromtxt('bias60vdata.csv', delimiter=",", usecols=(0))
data60by=np.genfromtxt('bias60vdata.csv', delimiter=",", usecols=(1))
```

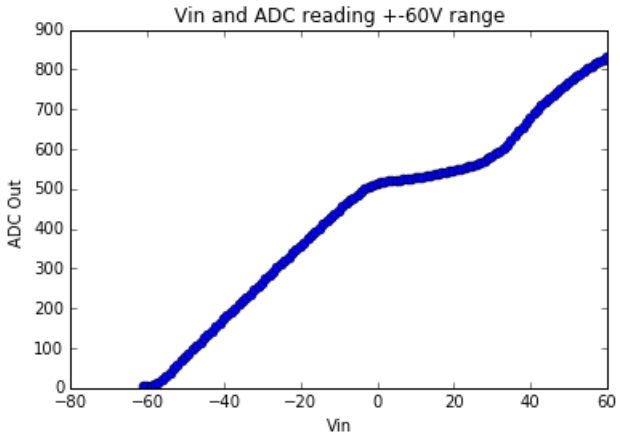
In [27]:
```
print data60bx
```

```
[-61. -60. -59. -58. -57. -56. -55. -54. -53. -52. -51. -50. -49. -4
8. -47.
 -46. -45. -44. -43. -42. -41. -40. -39. -38. -37. -36. -35. -34. -3
3. -32.
 -31. -30. -29. -28. -27. -26. -25. -24. -23. -22. -21. -20. -19. -1
8. -17.
 -16. -15. -14. -13. -12. -11. -10.  -9.  -8.  -7.  -6.  -5.  -4.  -
3.  -2.
  -1.   0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.  1
2.  13.
  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  2
7.  28.
  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  4
2.  43.
  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56.  5
7.  58.
  59.  60.]
```

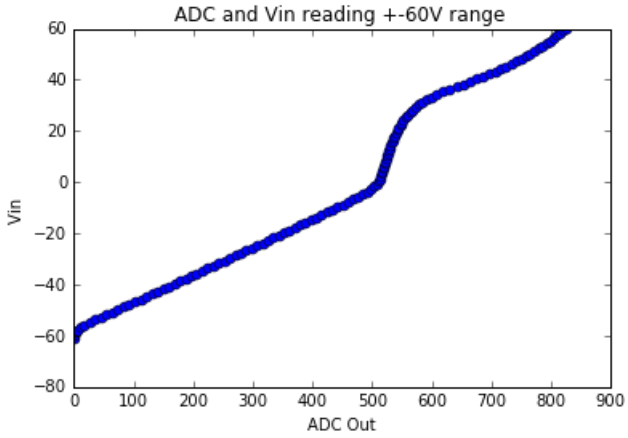
In [28]:
```
print data60by
```

```
[   3.    3.    4.    7.   12.   19.   28.   37.   47.   56.   66.
75.
   84.   94.  104.  114.  123.  133.  142.  152.  161.  171.  179.
189.
  198.  207.  217.  226.  235.  244.  254.  263.  272.  281.  290.
300.
  309.  318.  327.  336.  345.  354.  363.  373.  382.  390.  399.
409.
  417.  426.  435.  444.  453.  461.  469.  477.  485.  493.  499.
505.
  509.  513.  514.  516.  518.  519.  520.  521.  522.  524.  525.
526.
  528.  529.  531.  532.  534.  536.  538.  540.  542.  544.  546.
549.
  551.  554.  557.  561.  565.  569.  574.  580.  586.  593.  601.
610.
  620.  631.  643.  654.  666.  677.  688.  698.  708.  717.  726.
735.
  743.  751.  759.  766.  774.  781.  787.  794.  800.  806.  812.
817.
  823.  828.]
```

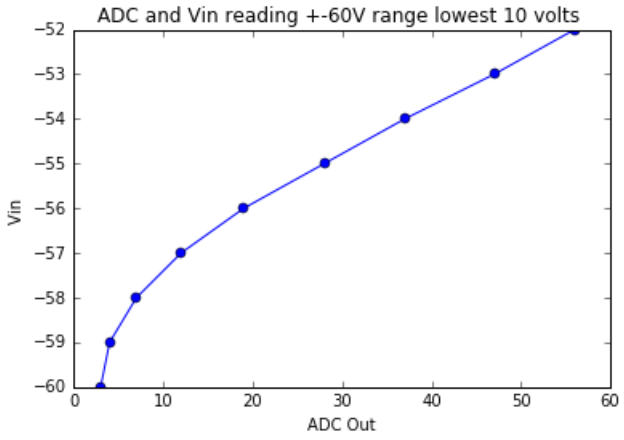
In [29]:
```
plt.title ('Vin and ADC reading +-60V range')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(data60bx, data60by, 'o-')

plt.show()
```



In [30]:
```
plt.title ('ADC and Vin reading +-60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60by, data60bx, 'o-')

plt.show()
```



In [31]:
```
plt.title ('ADC and Vin reading +-60V range lowest 10 volts ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60by[1:10], data60bx[1:10], 'o-')

plt.show()
```

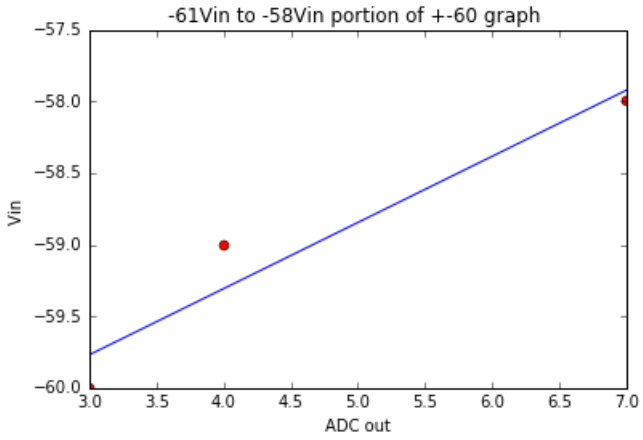


In [190]:
```
z9 = np.polyfit(data60by[1:4],data60bx[1:4],1)

print z9
```

```
[  0.46153846 -61.15384615]
```

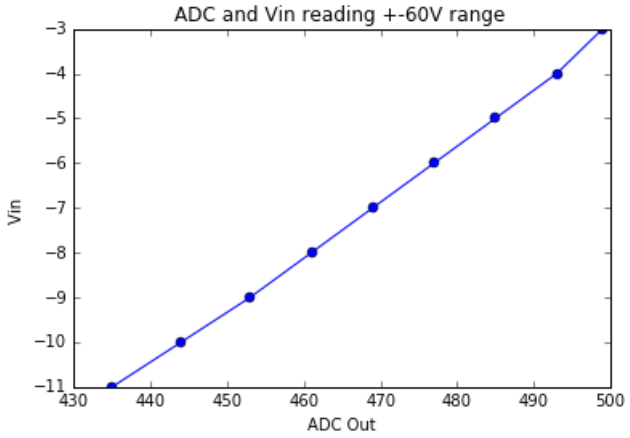
In [191]:
```
p60b6 = np.poly1d(z9)
```

In [192]:
```
plt.title ('-61Vin to -58Vin portion of +-60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[1:4], data60bx[1:4],'ro')
plt.plot(data60by[1:4], p60b6(data60by[1:4]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```



Because of this poor fitting, we will hard-code -60 through -57 vallues, and use other functions for the rest.

In [135]:
```
plt.title ('ADC and Vin reading +-60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60by[50:59], data60bx[50:59], 'o-')

plt.show()
```



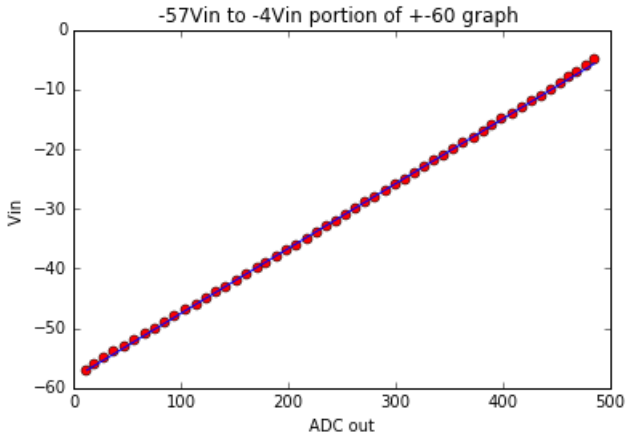
Based on the above plot, we have good linearity up until -3 Vin, so will use a linear function from Vin = -57 to Vin = -4.

In [136]:
```
z5 = np.polyfit(data60by[4:57],data60bx[4:57],1)

print z5
```

```
[  0.10863971 -58.35465865]
```

In [169]:
```
p60b1 = np.poly1d(z5)
```

In [170]:
```
plt.title ('-57Vin to -4Vin portion of +-60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[4:57], data60bx[4:57],'ro')
plt.plot(data60by[4:57], p60b1(data60by[4:57]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```

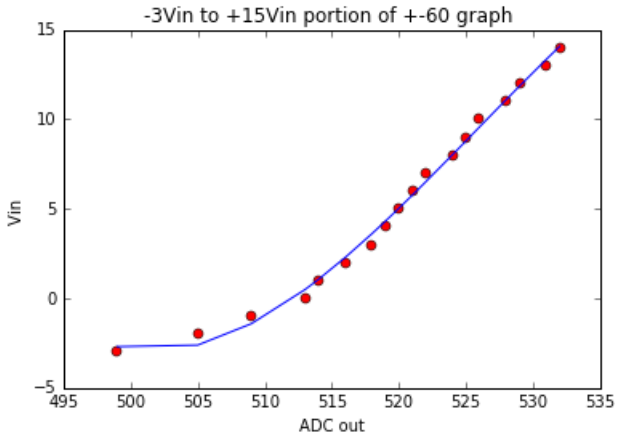


In [166]:
```
z6 = np.polyfit(data60by[58:76],data60bx[58:76],3)

print z6
```

```
[ -4.38444002e-04   6.91726325e-01  -3.63005678e+02   6.33739130e+04
]
```

In [171]:
```
p60b2 = np.poly1d(z6)
```

In [172]:
```
plt.title ('-3Vin to +15Vin portion of +-60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[58:76], data60bx[58:76],'ro')
plt.plot(data60by[58:76], p60b2(data60by[58:76]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```

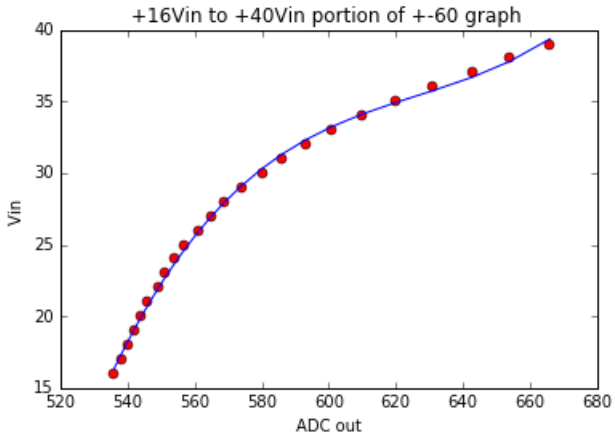


In [151]:
```
z7 = np.polyfit(data60by[77:101],data60bx[77:101],3)

print z7
```

```
[  1.66780325e-05  -3.13463796e-02   1.97126073e+01  -4.11219508e+03
]
```

In [173]:
```
p60b3 = np.poly1d(z7)
```

In [174]:
```
plt.title ('+16Vin to +40Vin portion of +-60 graph ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[77:101], data60bx[77:101],'ro')
plt.plot(data60by[77:101], p60b3(data60by[77:101]), '-')

#datax_os = np.linspace(np.min(datax), np.max(datax), 1024)
#plt.plot(datax_os, f(datax_os)) # 'smoother' line

plt.show()
```



In [161]:
```
z8 = np.polyfit(data60by[102:121],data60bx[102:121],2)

print z8
```

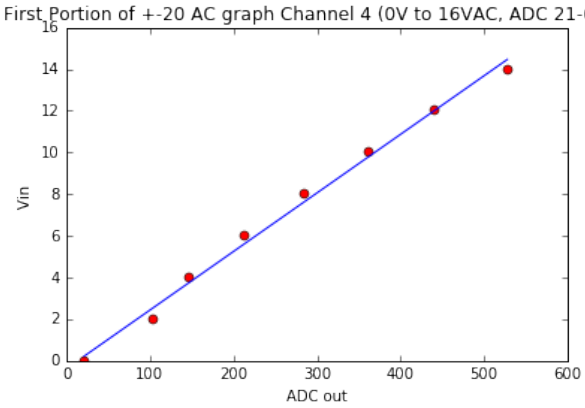
```
[  3.23432752e-04  -3.56694514e-01   1.33409998e+02]
```

In [175]:
```
p60b4 = np.poly1d(z8)
```

In [176]:
```
plt.title ('+41Vin to +60Vin portion of +-60 graph ')
```

In [398]:
```
plt.title ('First Portion of +-20 AC graph Channel 4 (0V to 16VAC, ADC
21-603) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420ac[0:8], x420ac[0:8],'ro')
plt.plot(y420ac[0:8], pz46(y420ac[0:8]), '-')
```

Out[398]:
```
[<matplotlib.lines.Line2D at 0x119d48e10>]
```



In [399]:
```
z47 = np.polyfit(y420ac[8:13],x420ac[8:13],1)

print z47
```

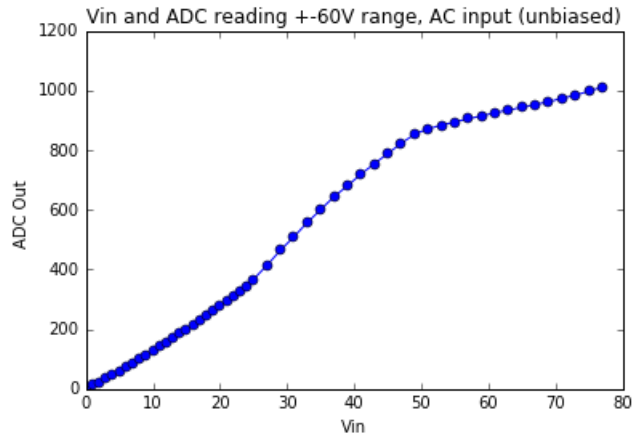
```
[  0.08866799 -37.45685885]
```

In [400]:
```
pz47 = np.poly1d(z47)
```

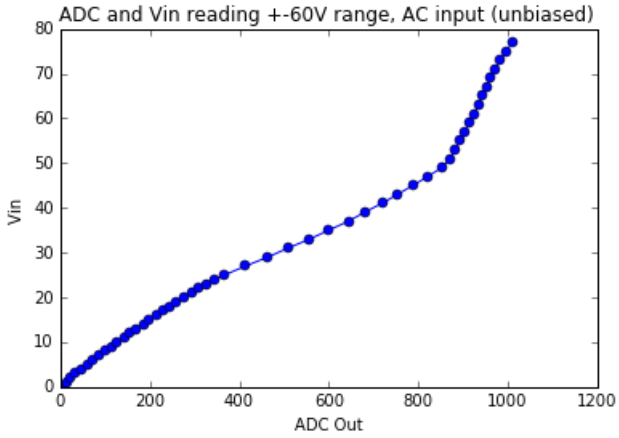
In [140]:
```
print data60acy
```

```
[    5.    15.    22.    34.    45.    60.    73.    86.   100.   11
4.
   127.   142.   155.   170.   186.   199.   215.   230.   245.   26
0.
   277.   294.   310.   328.   345.   367.   414.   464.   509.   55
7.
   600.   644.   680.   719.   754.   788.   822.   853.   871.   88
3.
   894.   905.   914.   925.   934.   943.   952.   962.   973.   98
4.
   997.  1011.]
```

In [142]:
```
plt.title ('Vin and ADC reading +-60V range, AC input (unbiased)')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(data60acx, data60acy, 'o-')

plt.show()
```



In [143]:
```
plt.title ('ADC and Vin reading +-60V range, AC input (unbiased) ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60acy, data60acx, 'o-')

plt.show()
```

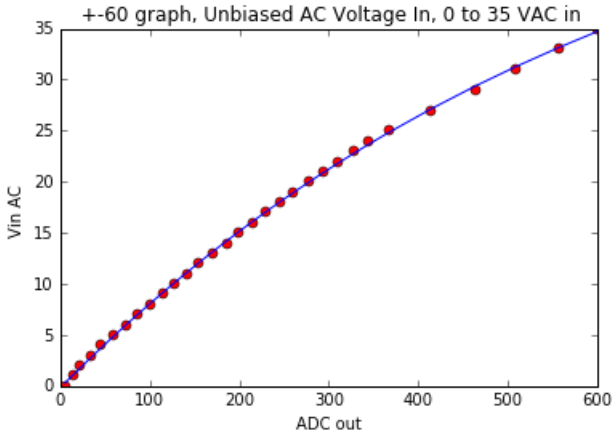


In [160]:
```
z10a = np.polyfit(data60acy[0:31],data60acx[0:31],3)

print z10a
```

```
[  1.97506257e-08  -6.17513960e-05   8.80140172e-02  -1.45966721e-01
]
```

In [152]:
```
p60ac1 = np.poly1d(z10a)
```

In [153]:
```
plt.title (' +-60 graph, Unbiased AC Voltage In, 0 to 35 VAC in')
plt.xlabel('ADC out')
plt.ylabel('Vin AC ')

plt.plot(data60acy[0:31], data60acx[0:31],'ro')
plt.plot(data60acy[0:31], p60ac1(data60acy[0:31]), '-')


plt.show()
```

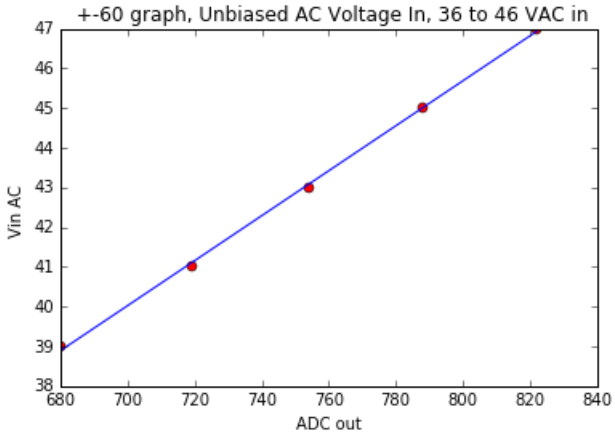


In [154]:
```
z10b = np.polyfit(data60acy[32:37],data60acx[32:37],1)

print z10b
```

```
[ 0.05661043  0.39499006]
```

In [155]:
```
p60ac2 = np.poly1d(z10b)
```

In [156]:
```
plt.title (' +-60 graph, Unbiased AC Voltage In, 36 to 46 VAC in')
plt.xlabel('ADC out')
plt.ylabel('Vin AC ')

plt.plot(data60acy[32:37], data60acx[32:37],'ro')
plt.plot(data60acy[32:37], p60ac2(data60acy[32:37]), '-')


plt.show()
```

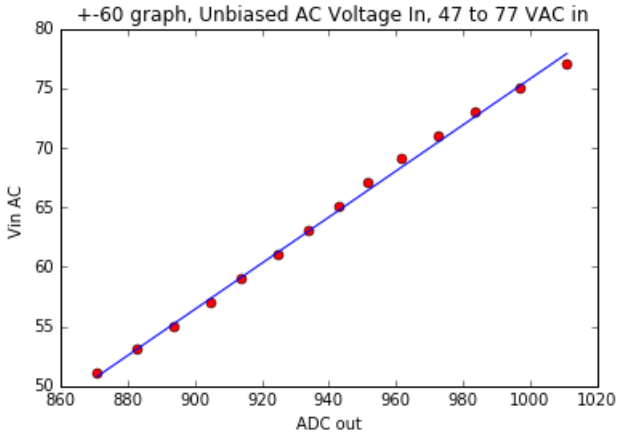


In [157]:
```
z10c = np.polyfit(data60acy[38:52],data60acx[38:52],1)

print z10c
```

```
[   0.19356398 -117.78423188]
```

In [158]:
```
p60ac3 = np.poly1d(z10c)
```

In [159]:
```
plt.title (' +-60 graph, Unbiased AC Voltage In, 47 to 77 VAC in')
plt.xlabel('ADC out')
plt.ylabel('Vin AC ')

plt.plot(data60acy[38:52], data60acx[38:52],'ro')
plt.plot(data60acy[38:52], p60ac3(data60acy[38:52]), '-')


plt.show()
```



In [126]:
```
data20acx=np.genfromtxt('AC20vdata.csv', delimiter=",", usecols=(0))
data20acy=np.genfromtxt('AC20vdata.csv', delimiter=",", usecols=(1))
```

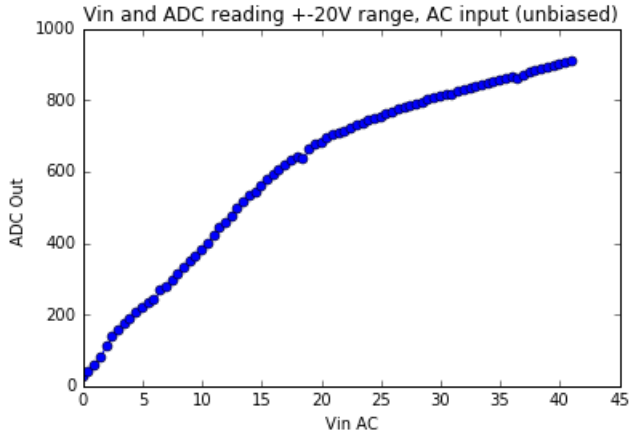
In [127]:
```
print data20acx
```

```
[  0.    0.5   1.    1.5   2.    2.5   3.    3.5   4.    4.5   5.
5.5
   6.    6.5   7.    7.5   8.    8.5   9.    9.5  10.   10.5  11.
11.5
  12.   12.5  13.   13.5  14.   14.5  15.   15.5  16.   16.5  17.
17.5
  18.   18.5  19.   19.5  20.   20.5  21.   21.5  22.   22.5  23.
23.5
  24.   24.5  25.   25.5  26.   26.5  27.   27.5  28.   28.5  29.
29.5
  30.   30.5  31.   31.5  32.   32.5  33.   33.5  34.   34.5  35.
35.5
  36.   36.5  37.   37.5  38.   38.5  39.   39.5  40.   40.5  41. ]
```

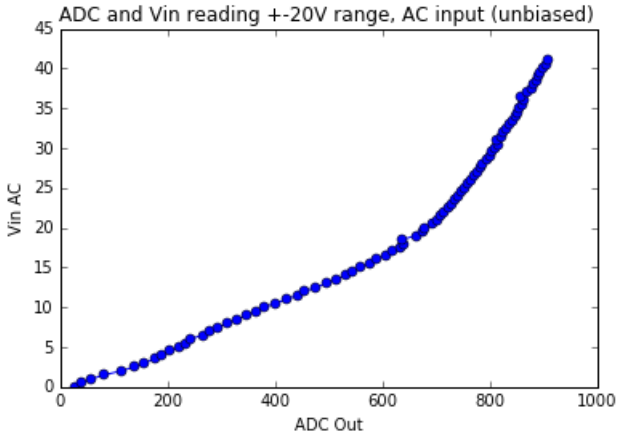
In [128]:
```
print data20acy
```

```
[  26.   39.   58.   81.  113.  137.  156.  175.  188.  204.  220.
233.
  241.  266.  279.  294.  312.  330.  348.  364.  380.  400.  422.
441.
  455.  476.  496.  513.  532.  543.  559.  577.  589.  605.  617.
632.
  639.  635.  664.  675.  678.  693.  702.  707.  713.  722.  730.
735.
  742.  748.  754.  760.  765.  772.  777.  782.  787.  794.  800.
805.
  809.  815.  813.  822.  826.  831.  837.  841.  847.  851.  855.
859.
  864.  858.  870.  877.  880.  886.  890.  894.  899.  904.  908.]
```

In [130]:
```
plt.title ('Vin and ADC reading +-20V range, AC input (unbiased)')
plt.xlabel('Vin AC')
plt.ylabel('ADC Out ')

plt.plot(data20acx, data20acy, 'o-')

plt.show()
```



In [131]:
```
plt.title ('ADC and Vin reading +-20V range, AC input (unbiased) ')
plt.xlabel('ADC Out')
plt.ylabel('Vin AC')

plt.plot(data20acy, data20acx, 'o-')

plt.show()
```

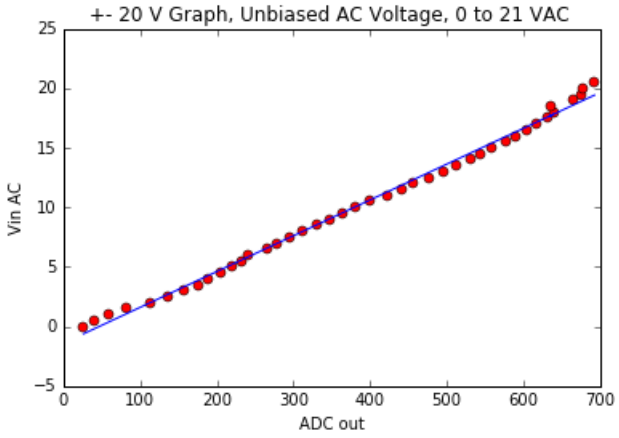


In [132]:
```
z11 = np.polyfit(data20acy[0:42],data20acx[0:42],1)

print z11
```

```
[ 0.0300406 -1.3978863]
```

In [133]:
```
p20ac1 = np.poly1d(z11)
```

In [134]:
```
plt.title ('+- 20 V Graph, Unbiased AC Voltage, 0 to 21 VAC ')
plt.xlabel('ADC out')
plt.ylabel('Vin AC')

plt.plot(data20acy[0:42], data20acx[0:42],'ro')
plt.plot(data20acy[0:42], p20ac1(data20acy[0:42]), '-')


plt.show()
```

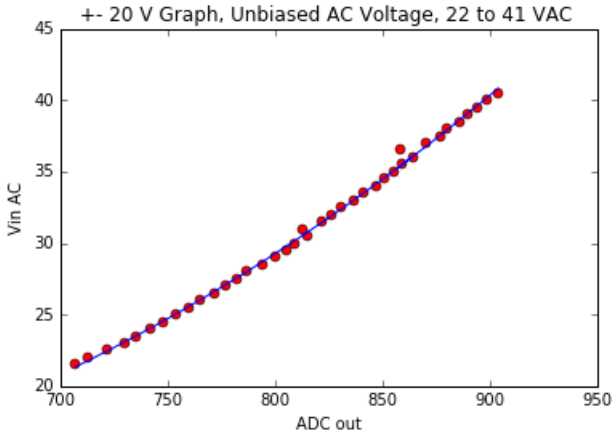


In [135]:
```
z11b = np.polyfit(data20acy[43:82],data20acx[43:82],2)

print z11b
```

```
[  1.32483337e-04  -1.14589298e-01   3.61364727e+01]
```

In [136]:
```
p20ac1b = np.poly1d(z11b)
```

In [137]:
```
plt.title ('+- 20 V Graph, Unbiased AC Voltage, 22 to 41 VAC ')
plt.xlabel('ADC out')
plt.ylabel('Vin AC')

plt.plot(data20acy[43:82], data20acx[43:82],'ro')
plt.plot(data20acy[43:82], p20ac1b(data20acy[43:82]), '-')


plt.show()
```



In [57]:
```
data20bx=np.genfromtxt('bias20vdata.csv', delimiter=",", usecols=(0))
data20by=np.genfromtxt('bias20vdata.csv', delimiter=",", usecols=(1))
```

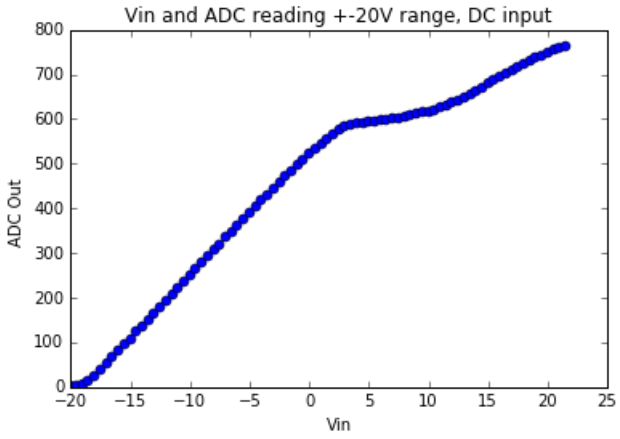
In [58]:
```
print data20bx
```

```
[-20.  -19.5 -19.  -18.5 -18.  -17.5 -17.  -16.5 -16.  -15.5 -15.  -
14.5
 -14.  -13.5 -13.  -12.5 -12.  -11.5 -11.  -10.5 -10.   -9.5  -9.
-8.5
  -8.   -7.5  -7.   -6.5  -6.   -5.5  -5.   -4.5  -4.   -3.5  -3.
-2.5
  -2.   -1.5  -1.   -0.5   0.    0.5   1.    1.5   2.    2.5   3.
3.5
   4.    4.5   5.    5.5   6.    6.5   7.    7.5   8.    8.5   9.
9.5
  10.   10.5  11.   11.5  12.   12.5  13.   13.5  14.   14.5  15.
15.5
  16.   16.5  17.   17.5  18.   18.5  19.   19.5  20.   20.5  21.
21.5]
```

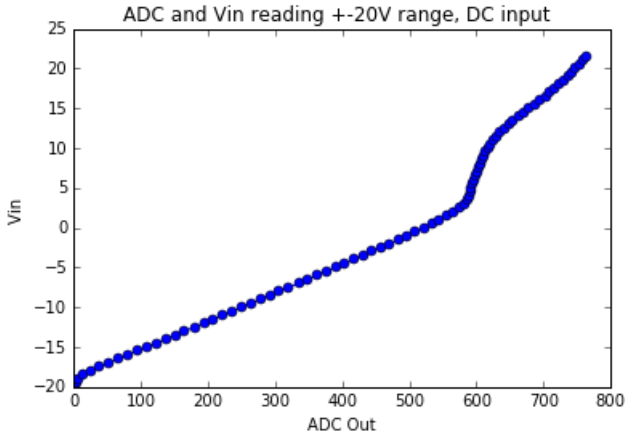
In [59]:
```
print data20by
```

```
[   2.    3.    6.   13.   25.   38.   52.   66.   80.   95.  108.
123.
  137.  151.  165.  180.  194.  207.  222.  236.  250.  264.  278.
292.
  306.  320.  335.  348.  362.  376.  390.  404.  418.  431.  444.
458.
  471.  484.  497.  509.  522.  534.  545.  556.  566.  575.  582.
586.
  589.  591.  593.  595.  597.  599.  601.  603.  605.  608.  611.
614.
  617.  621.  625.  629.  636.  641.  648.  655.  663.  671.  679.
687.
  695.  703.  710.  717.  724.  730.  737.  742.  748.  754.  758.
764.]
```

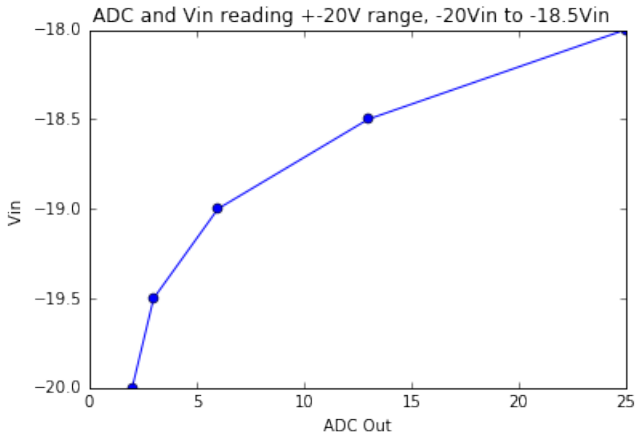
In [60]:
```
plt.title ('Vin and ADC reading +-20V range, DC input')
plt.xlabel('Vin')
plt.ylabel('ADC Out ')

plt.plot(data20bx, data20by, 'o-')

plt.show()
```



In [61]:
```
plt.title ('ADC and Vin reading +-20V range, DC input  ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data20by, data20bx, 'o-')

plt.show()
```



In [114]:
```
plt.title ('ADC and Vin reading +-20V range, -20Vin to -18.5Vin  ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data20by[0:5], data20bx[0:5], 'o-')

plt.show()
```

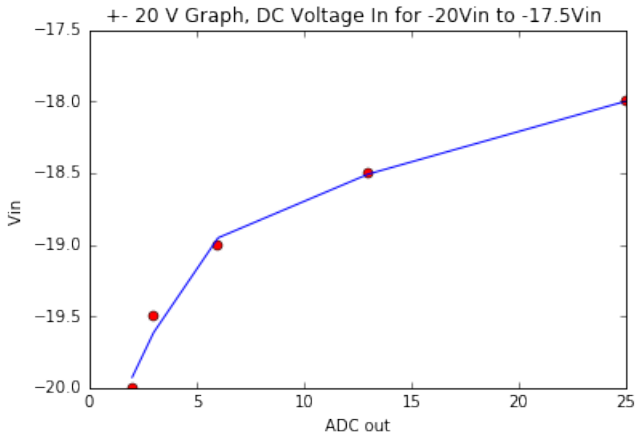


In [121]:
```
z12 = np.polyfit(data20by[0:5],data20bx[0:5],3)

print z12
```

```
[  6.61743137e-04  -3.02298547e-02   4.50641886e-01  -2.07114737e+01
]
```

In [122]:
```
p20b1 = np.poly1d(z12)
```

In [123]:
```
plt.title ('+- 20 V Graph, DC Voltage In for -20Vin to -17.5Vin ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[0:5], data20bx[0:5],'ro')
plt.plot(data20by[0:5], p20b1(data20by[0:5]), '-')


plt.show()
```

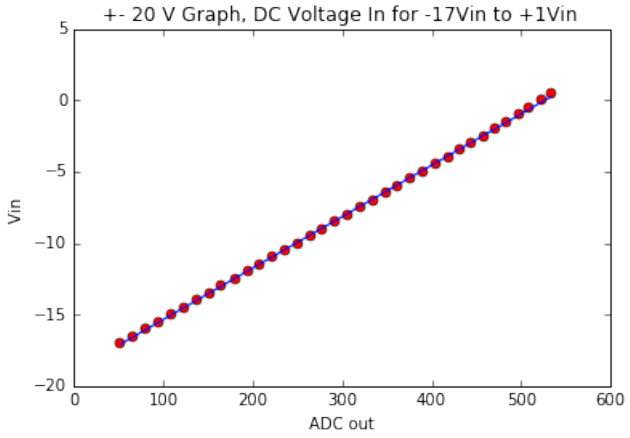


In [108]:
```
z13 = np.polyfit(data20by[6:42],data20bx[6:42],1)

print z13
```

```
[  0.03595532 -18.94238609]
```

In [109]:
```
p20b2 = np.poly1d(z13)
```

In [125]:
```
plt.title ('+- 20 V Graph, DC Voltage In for -17Vin to +1Vin ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[6:42], data20bx[6:42],'ro')
plt.plot(data20by[6:42], p20b2(data20by[6:42]), '-')


plt.show()
```

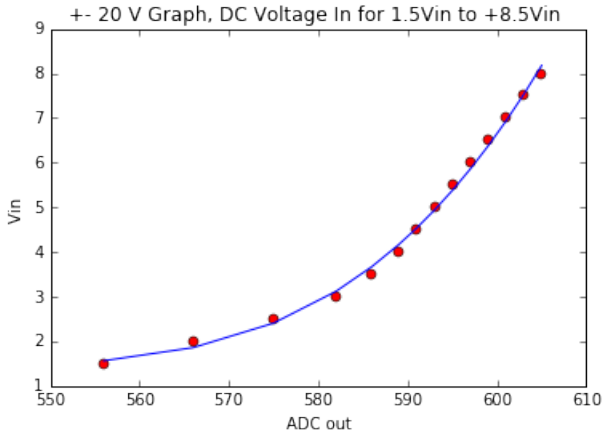


In [91]:
```
z14 = np.polyfit(data20by[43:57],data20bx[43:57],3)

print z14
```

```
[  3.49257851e-05  -5.76113098e-02   3.16927686e+01  -5.81291422e+03
]
```

In [92]:
```
p20b3 = np.poly1d(z14)
```

In [94]:
```
plt.title ('+- 20 V Graph, DC Voltage In for 1.5Vin to +8.5Vin ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[43:57], data20bx[43:57],'ro')
plt.plot(data20by[43:57], p20b3(data20by[43:57]), '-')


plt.show()
```

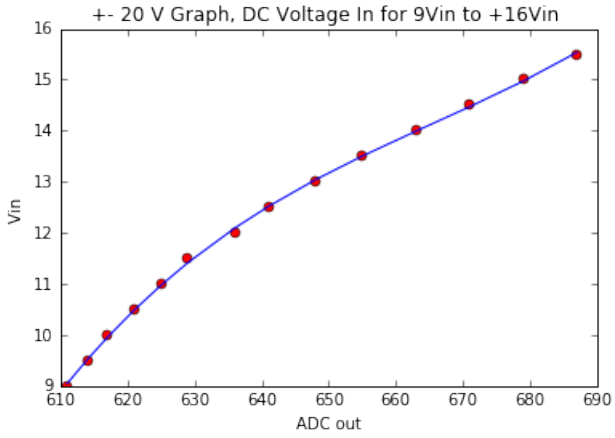


In [95]:
```
z15 = np.polyfit(data20by[58:72],data20bx[58:72],3)

print z15
```

```
[  1.15042808e-05  -2.29645386e-02   1.53398426e+01  -3.41456785e+03
]
```

In [96]:
```
p20b4 = np.poly1d(z15)
```

In [97]:
```
plt.title ('+- 20 V Graph, DC Voltage In for 9Vin to +16Vin ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[58:72], data20bx[58:72],'ro')
plt.plot(data20by[58:72], p20b4(data20by[58:72]), '-')


plt.show()
```

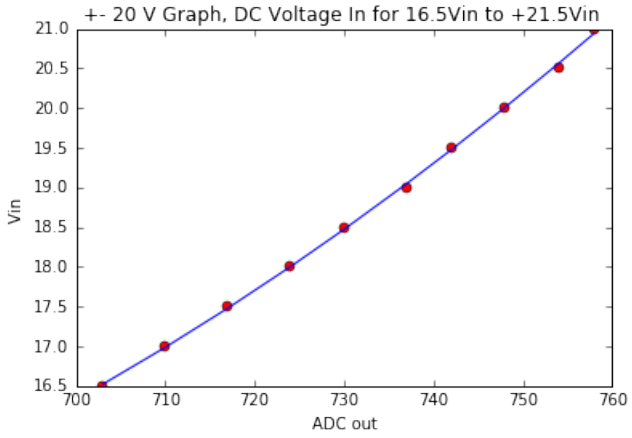


In [98]:
```
z16 = np.polyfit(data20by[73:83],data20bx[73:83],2)

print z16
```

```
[  2.90726437e-04  -3.44365552e-01   1.14931164e+02]
```

In [99]:
```
p20b5 = np.poly1d(z16)
```

In [100]:
```
plt.title ('+- 20 V Graph, DC Voltage In for 16.5Vin to +21.5Vin ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[73:83], data20bx[73:83],'ro')
plt.plot(data20by[73:83], p20b5(data20by[73:83]), '-')


plt.show()
```



In [ ]:

In [2]:
```
import matplotlib.pyplot as plt
import csv
import numpy as np
from numpy import arange,array,ones,linalg
from pylab import plot,show
from scipy.optimize import curve_fit
%matplotlib inline
```

In [422]:
```
# datax=np.genfromtxt('60vdata.csv', delimiter=",")
datax=np.genfromtxt('Ch260vdata.csv', delimiter=",", usecols=(0))
datay=np.genfromtxt('Ch260vdata.csv', delimiter=",", usecols=(1))
```

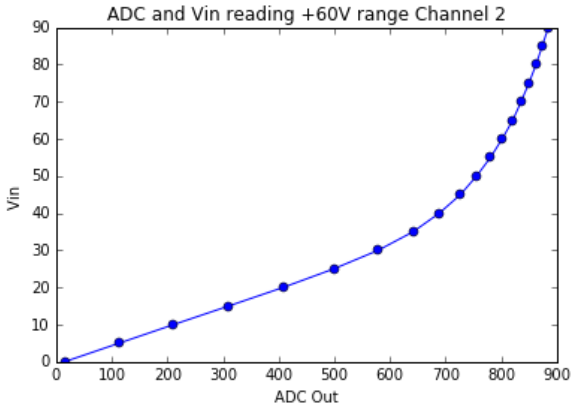
In [423]:
```
print datax
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.  85.  90.]
```

In [424]:
```
print datay
```

```
[  15.  113.  211.  310.  408.  499.  579.  642.  689.  727.  756.
781.
  802.  821.  836.  850.  863.  874.  885.]
```

In [425]:
```
plt.title ('ADC and Vin reading +60V range Channel 2')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(datay, datax, 'o-')

plt.show()
```



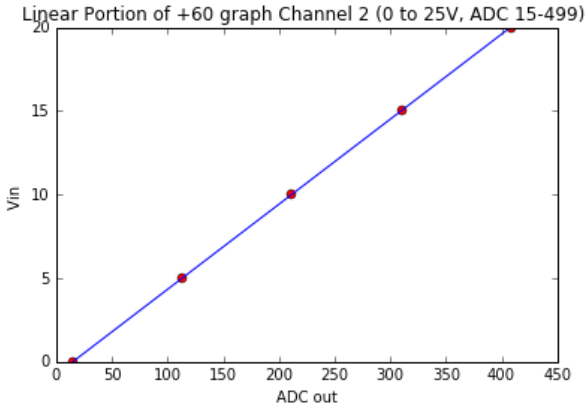
In [426]:
```
z1 = np.polyfit(datay[0:5],datax[0:5],1)

print z1
```

```
[ 0.05086454 -0.75276417]
```

In [427]:
```
pz1 = np.poly1d(z1)
```

In [428]:
```
plt.title ('Linear Portion of +60 graph Channel 2 (0 to 25V, ADC 15-49
9) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(datay[0:5], datax[0:5],'ro')
plt.plot(datay[0:5], pz1(datay[0:5]), '-')
```

Out[428]:
```
[<matplotlib.lines.Line2D at 0x11bb22990>]
```

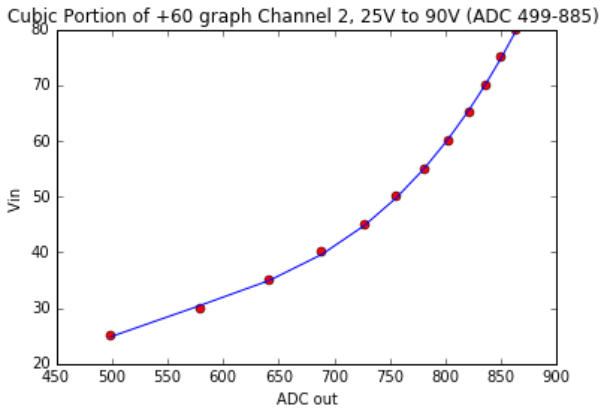


In [431]:
```
z2 = np.polyfit(datay[5:17],datax[5:17],3)

print z2
```

```
[  1.21032951e-06  -2.06514637e-03   1.23923536e+00  -2.29692002e+02
]
```

In [432]:
```
pz2 = np.poly1d(z2)
```

In [433]:
```
plt.title ('Cubic Portion of +60 graph Channel 2, 25V to 90V (ADC 499-885) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(datay[5:17], datax[5:17],'ro')
plt.plot(datay[5:17], pz2(datay[5:17]), '-')

plt.show()
```



In [434]:
```
# datax=np.genfromtxt('60vdata.csv', delimiter=",")
data20x=np.genfromtxt('Ch220vdata.csv', delimiter=",", usecols=(0))
data20y=np.genfromtxt('Ch220vdata.csv', delimiter=",", usecols=(1))
```

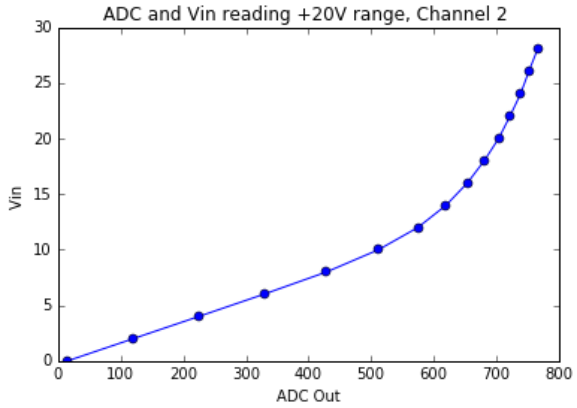
In [435]:
```
print data20x
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.  28.]
```

In [436]:
```
print data20y
```

```
[  15.  120.  225.  329.  429.  512.  575.  620.  654.  681.  704.
722.
  739.  752.  766.]
```

In [437]:
```
plt.title ('ADC and Vin reading +20V range, Channel 2 ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data20y, data20x, 'o-')

plt.show()
```



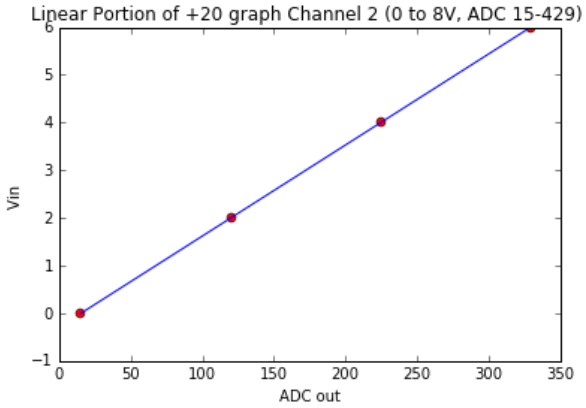
In [438]:
```
z3 = np.polyfit(data20y[0:4],data20x[0:4],1)

print z3
```

```
[ 0.01910209 -0.29033538]
```

In [439]:
```
pz3 = np.poly1d(z3)
```

In [440]:
```
plt.title ('Linear Portion of +20 graph Channel 2 (0 to 8V, ADC 15-429
) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20y[0:4], data20x[0:4],'ro')
plt.plot(data20y[0:4], pz3(data20y[0:4]), '-')
```

Out[440]:
```
[<matplotlib.lines.Line2D at 0x11c88acd0>]
```

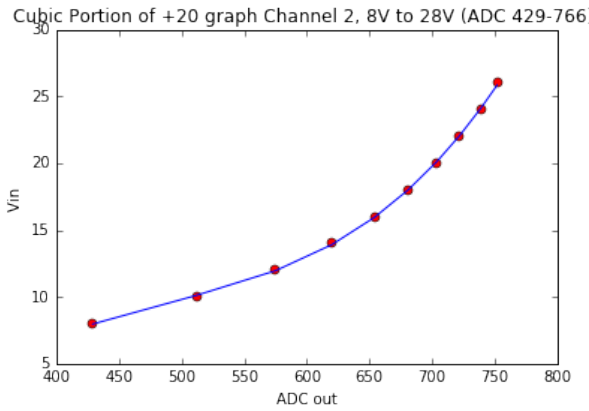


In [441]:
```
z4 = np.polyfit(data20y[4:14],data20x[4:14],3)

print z4
```

```
[  5.64814913e-07  -8.33633285e-04   4.34261015e-01  -6.95018065e+01
]
```

In [442]:
```
pz4 = np.poly1d(z4)
```

In [443]:
```
plt.title ('Cubic Portion of +20 graph Channel 2, 8V to 28V (ADC 429-7
66) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20y[4:14], data20x[4:14],'ro')
plt.plot(data20y[4:14], pz4(data20y[4:14]), '-')

plt.show()
```



In [444]:
```
data60bx=np.genfromtxt('Ch2bias60vdata.csv', delimiter=",", usecols=(0
))
data60by=np.genfromtxt('Ch2bias60vdata.csv', delimiter=",", usecols=(1
))
```

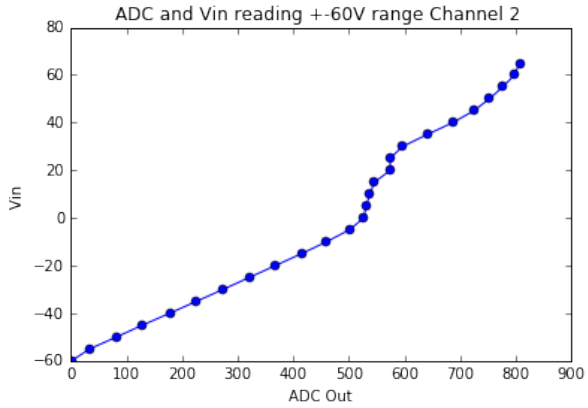
In [445]:
```
print data60bx
```

```
[-60. -55. -50. -45. -40. -35. -30. -25. -20. -15. -10.  -5.   0.
5.  10.
  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  65.]
```

In [446]:
```
print data60by
```

```
[   1.   34.   82.  129.  178.  226.  274.  321.  368.  415.  460.
501.
  527.  531.  538.  546.  575.  574.  597.  641.  687.  725.  753.
777.
  798.  807.]
```

In [447]:
```
plt.title ('ADC and Vin reading +-60V range Channel 2 ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60by, data60bx, 'o-')

plt.show()
```



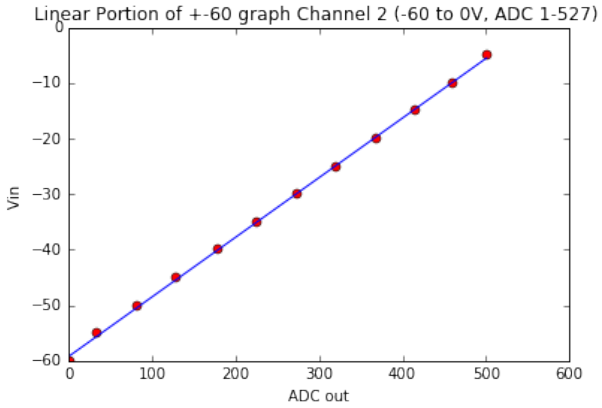
In [450]:
```
z5 = np.polyfit(data60by[0:12],data60bx[0:12],1)

print z5
```

```
[  0.10716274 -59.19245342]
```

In [451]:
```
pz5 = np.poly1d(z5)
```

In [453]:
```
plt.title ('Linear Portion of +-60 graph Channel 2 (-60 to 0V, ADC 1-5
27) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[0:12], data60bx[0:12],'ro')
plt.plot(data60by[0:12], pz5(data60by[0:12]), '-')
```

Out[453]:
```
[<matplotlib.lines.Line2D at 0x11cf89390>]
```



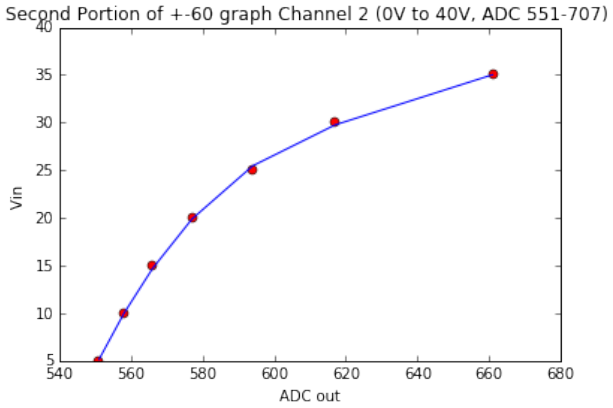
In [34]:
```
z6 = np.polyfit(data60by[13:20],data60bx[13:20],3)

print z6
```

```
[  3.03090955e-05  -5.77381309e-02   3.67667665e+01  -7.79421689e+03
]
```

In [35]:
```
pz6 = np.poly1d(z6)
```

In [39]:
```
plt.title ('Second Portion of +-60 graph Channel 2 (0V to 40V, ADC 551
-707) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[13:20], data60bx[13:20],'ro')
plt.plot(data60by[13:20], pz6(data60by[13:20]), '-')
```

Out[39]:
```
[<matplotlib.lines.Line2D at 0x10d921b50>]
```



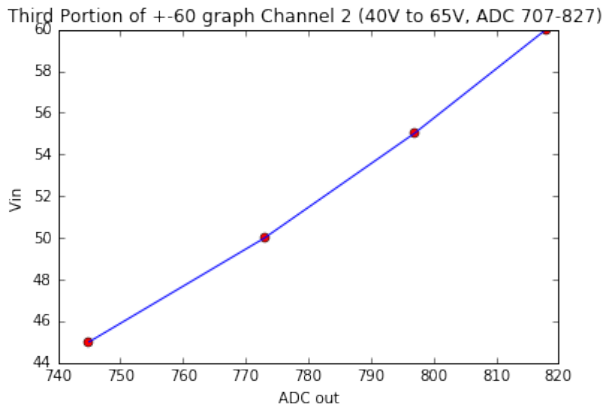
In [41]:
```
z7 = np.polyfit(data60by[21:25],data60bx[21:25],2)

print z7
```

```
[  6.09276138e-04  -7.46959375e-01   2.63325274e+02]
```

In [42]:
```
pz7 = np.poly1d(z7)
```

In [43]:
```
plt.title ('Third Portion of +-60 graph Channel 2 (40V to 65V, ADC 707
-827) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60by[21:25], data60bx[21:25],'ro')
plt.plot(data60by[21:25], pz7(data60by[21:25]), '-')
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x10dd43c50>]
```



In [454]:
```
data20bx=np.genfromtxt('Ch2bias20vdata.csv', delimiter=",", usecols=(0
))
data20by=np.genfromtxt('Ch2bias20vdata.csv', delimiter=",", usecols=(1
))
```

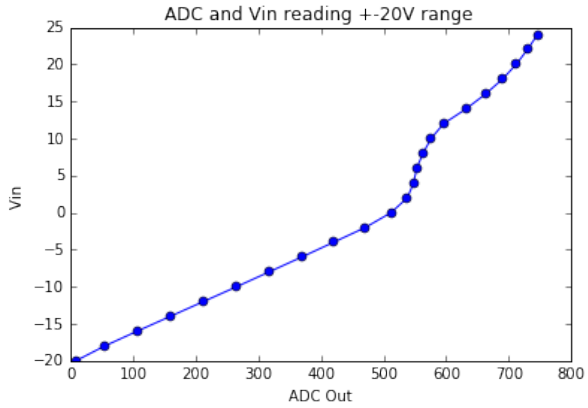
In [455]:
```
print data20bx
```

```
[-20. -18. -16. -14. -12. -10.  -8.  -6.  -4.  -2.   0.   2.   4.
6.   8.
  10.  12.  14.  16.  18.  20.  22.  24.]
```

In [456]:
```
print data20by
```

```
[   8.   54.  106.  159.  212.  265.  317.  369.  420.  471.  512.
538.
  548.  553.  563.  576.  596.  632.  663.  690.  712.  730.  746.]
```

In [457]:
```
plt.title ('ADC and Vin reading +-20V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data20by, data20bx, 'o-')

plt.show()
```



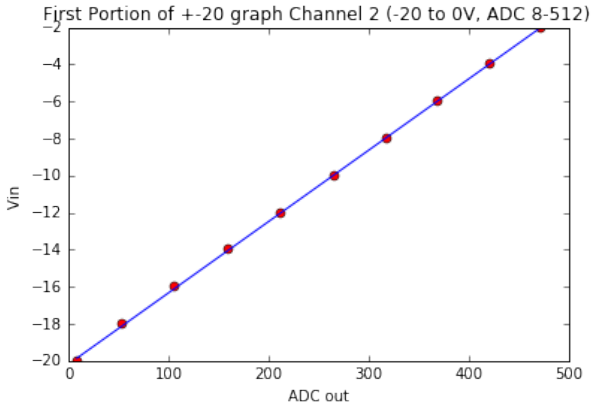
In [459]:
```
z8 = np.polyfit(data20by[0:10],data20bx[0:10],1)

print z8
```

```
[  0.03849697 -20.16612851]
```

In [460]:
```
pz8 = np.poly1d(z8)
```

In [461]:
```
plt.title ('First Portion of +-20 graph Channel 2 (-20 to 0V, ADC 8-51
2) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[0:10], data20bx[0:10],'ro')
plt.plot(data20by[0:10], pz8(data20by[0:10]), '-')
```

Out[461]:
```
[<matplotlib.lines.Line2D at 0x11dc9df10>]
```



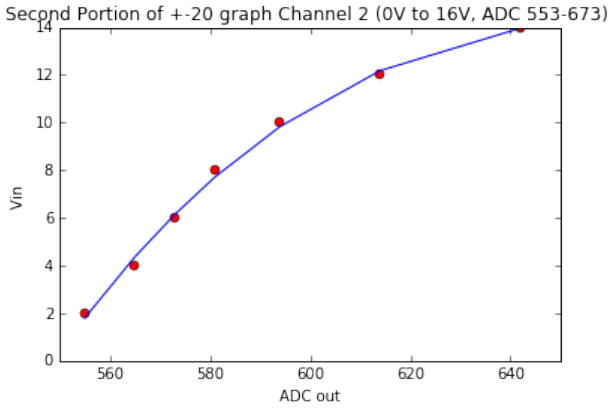
In [51]:
```
z9 = np.polyfit(data20by[11:18],data20bx[11:18],3)

print z9
```

```
[  4.48749242e-06  -9.41783264e-03   6.58221160e+00  -1.51755640e+03
]
```

In [52]:
```
pz9 = np.poly1d(z9)
```

In [53]:
```
plt.title ('Second Portion of +-20 graph Channel 2 (0V to 16V, ADC 553
-673) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[11:18], data20bx[11:18],'ro')
plt.plot(data20by[11:18], pz9(data20by[11:18]), '-')
```

Out[53]:
```
[<matplotlib.lines.Line2D at 0x10e46e790>]
```



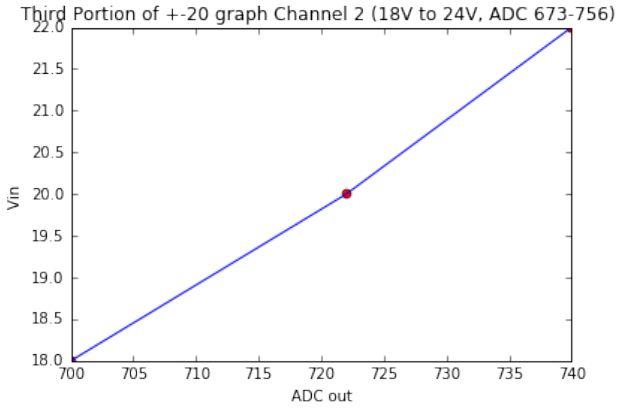
In [54]:
```
z10 = np.polyfit(data20by[19:22],data20bx[19:22],2)

print z10
```

```
[  5.05050505e-04  -6.27272727e-01   2.09616162e+02]
```

In [55]:
```
pz10 = np.poly1d(z10)
```

In [56]:
```
plt.title ('Third Portion of +-20 graph Channel 2 (18V to 24V, ADC 673
-756) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20by[19:22], data20bx[19:22],'ro')
plt.plot(data20by[19:22], pz10(data20by[19:22]), '-')
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0x10e6f1410>]
```



In [57]:
```
data60acx=np.genfromtxt('Ch2AC60vdata.csv', delimiter=",", usecols=(0)
)
data60acy=np.genfromtxt('Ch2AC60vdata.csv', delimiter=",", usecols=(1)
)
```

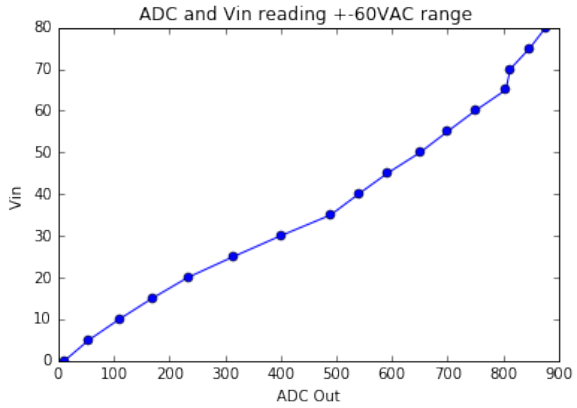
In [58]:
```
print data60acx
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.]
```

In [59]:
```
print data60acy
```

```
[  11.   55.  110.  169.  233.  315.  400.  490.  540.  592.  651.
700.
  750.  805.  813.  848.  877.]
```

In [60]:
```
plt.title ('ADC and Vin reading +-60VAC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data60acy, data60acx, 'o-')

plt.show()
```



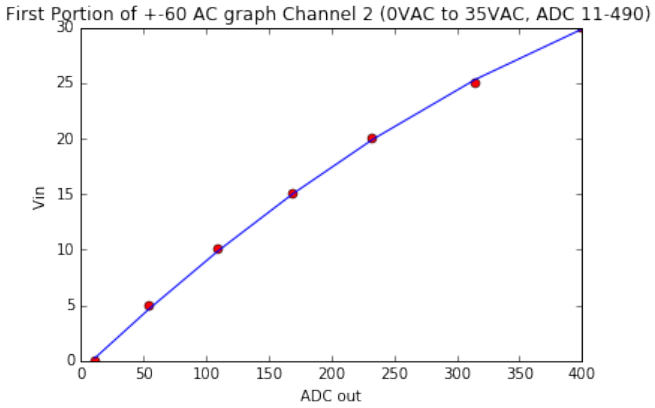
In [61]:
```
z11 = np.polyfit(data60acy[0:7],data60acx[0:7],2)

print z11
```

```
[ -7.53880621e-05   1.07066736e-01  -9.28507502e-01]
```

In [62]:
```
pz11 = np.poly1d(z11)
```

In [63]:
```
plt.title ('First Portion of +-60 AC graph Channel 2 (0VAC to 35VAC, A
DC 11-490) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60acy[0:7], data60acx[0:7],'ro')
plt.plot(data60acy[0:7], pz11(data60acy[0:7]), '-')
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x10d985490>]
```



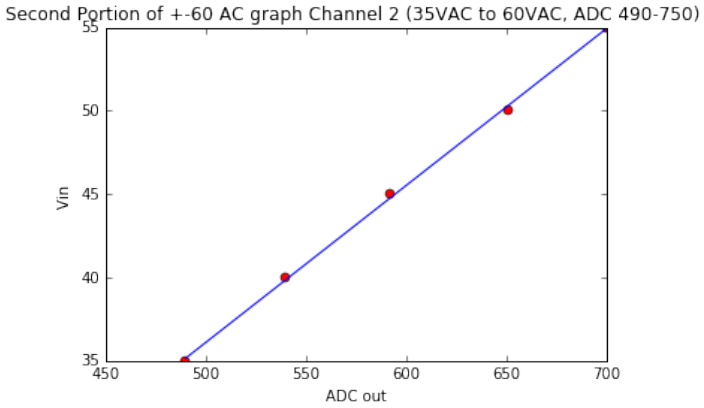
In [64]:
```
z12 = np.polyfit(data60acy[7:12],data60acx[7:12],1)

print z12
```

```
[  0.09408488 -10.94286869]
```

In [65]:
```
pz12 = np.poly1d(z12)
```

In [66]:
```
plt.title ('Second Portion of +-60 AC graph Channel 2 (35VAC to 60VAC,
ADC 490-750) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data60acy[7:12], data60acx[7:12],'ro')
plt.plot(data60acy[7:12], pz12(data60acy[7:12]), '-')
```

Out[66]:
```
[<matplotlib.lines.Line2D at 0x10ce56050>]
```



In [67]:
```
data20acx=np.genfromtxt('Ch2AC20vdata.csv', delimiter=",", usecols=(0)
)
data20acy=np.genfromtxt('Ch2AC20vdata.csv', delimiter=",", usecols=(1)
)
```

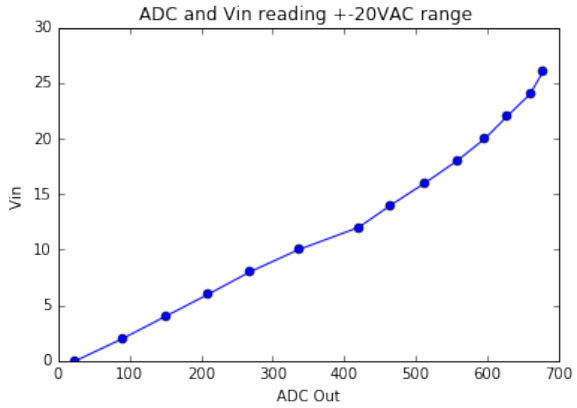
In [68]:
```
print data20acx
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.]
```

In [69]:
```
print data20acy
```

```
[  24.   90.  150.  210.  268.  336.  420.  465.  513.  558.  597.
628.
  660.  678.]
```

In [70]:
```
plt.title ('ADC and Vin reading +-20VAC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(data20acy, data20acx, 'o-')

plt.show()
```



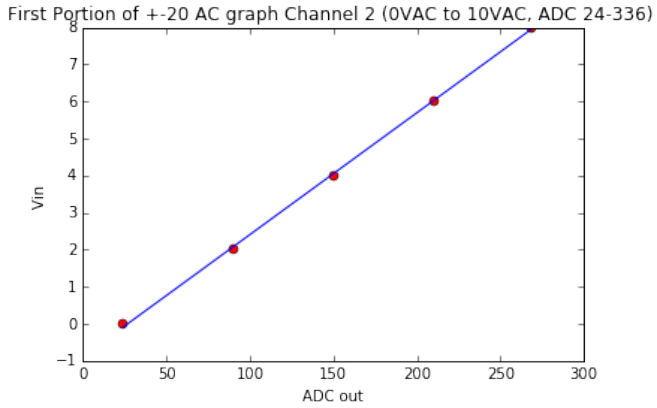
In [71]:
```
z13 = np.polyfit(data20acy[0:5],data20acx[0:5],1)

print z13
```

```
[ 0.03287624 -0.87883376]
```

In [72]:
```
pz13 = np.poly1d(z13)
```

In [73]:
```
plt.title ('First Portion of +-20 AC graph Channel 2 (0VAC to 10VAC, A
DC 24-336) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20acy[0:5], data20acx[0:5],'ro')
plt.plot(data20acy[0:5], pz13(data20acy[0:5]), '-')
```

Out[73]:
```
[<matplotlib.lines.Line2D at 0x10e927150>]
```



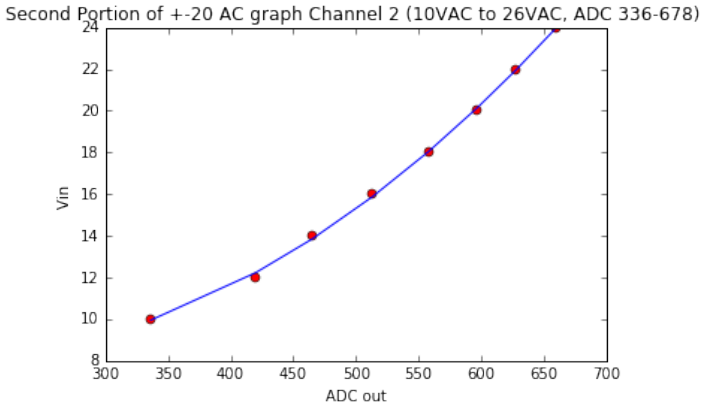
In [74]:
```
z14 = np.polyfit(data20acy[5:13],data20acx[5:13],2)

print z14
```

```
[  6.71253055e-05  -2.35743774e-02   1.02917784e+01]
```

In [75]:
```
pz14 = np.poly1d(z14)
```

In [76]:
```
plt.title ('Second Portion of +-20 AC graph Channel 2 (10VAC to 26VAC,
ADC 336-678) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(data20acy[5:13], data20acx[5:13],'ro')
plt.plot(data20acy[5:13], pz14(data20acy[5:13]), '-')
```

Out[76]:
```
[<matplotlib.lines.Line2D at 0x10ea81190>]
```



In [219]:
```
x360=np.genfromtxt('Ch360vdata.csv', delimiter=",", usecols=(0))
y360=np.genfromtxt('Ch360vdata.csv', delimiter=",", usecols=(1))
```

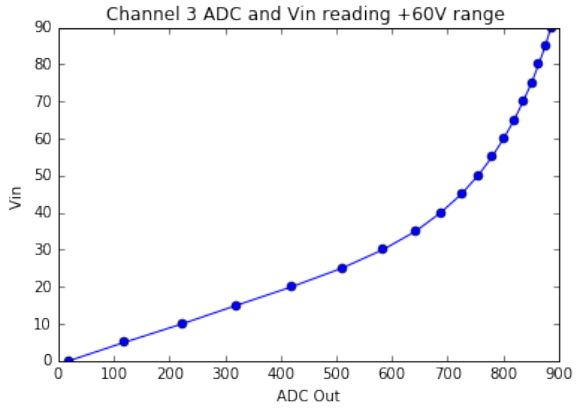
In [220]:
```
print x360
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.  85.  90.]
```

In [221]:
```
print y360
```

```
[  19.  119.  223.  321.  420.  510.  584.  643.  688.  725.  755.
780.
  801.  820.  836.  851.  864.  876.  886.]
```

In [222]:
```
plt.title ('Channel 3 ADC and Vin reading +60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y360, x360, 'o-')

plt.show()
```



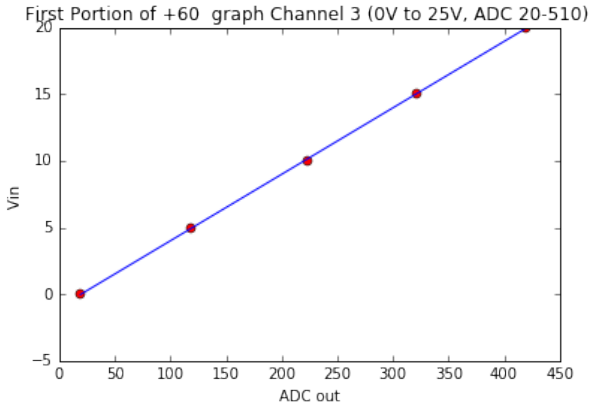
In [223]:
```
z15 = np.polyfit(y360[0:5],x360[0:5],1)

print z15
```

```
[ 0.04979605 -0.97505039]
```

In [224]:
```
pz15 = np.poly1d(z15)
```

In [225]:
```
plt.title ('First Portion of +60  graph Channel 3 (0V to 25V, ADC 20-5
10) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360[0:5], x360[0:5],'ro')
plt.plot(y360[0:5], pz15(y360[0:5]), '-')
```

Out[225]:
```
[<matplotlib.lines.Line2D at 0x1146675d0>]
```



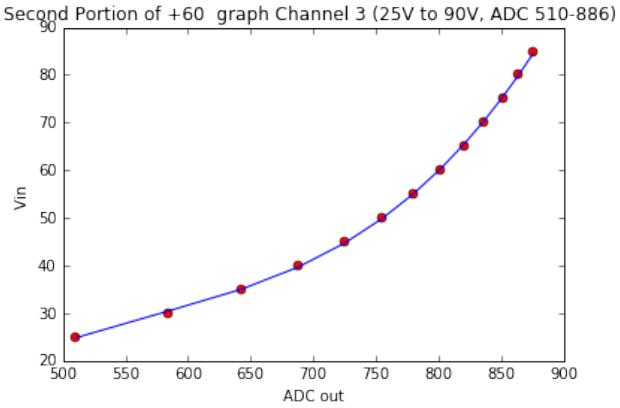
In [226]:
```
z16 = np.polyfit(y360[5:18],x360[5:18],3)

print z16
```

```
[  1.20630285e-06  -2.07707165e-03   1.26377865e+00  -2.39473546e+02
]
```

In [228]:
```
pz16 = np.poly1d(z16)
```

In [229]:
```
plt.title ('Second Portion of +60  graph Channel 3 (25V to 90V, ADC 51
0-886) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360[5:18], x360[5:18],'ro')
plt.plot(y360[5:18], pz16(y360[5:18]), '-')
```

Out[229]:
```
[<matplotlib.lines.Line2D at 0x1147b7ed0>]
```



In [209]:
```
x320=np.genfromtxt('Ch320vdata.csv', delimiter=",", usecols=(0))
y320=np.genfromtxt('Ch320vdata.csv', delimiter=",", usecols=(1))
```

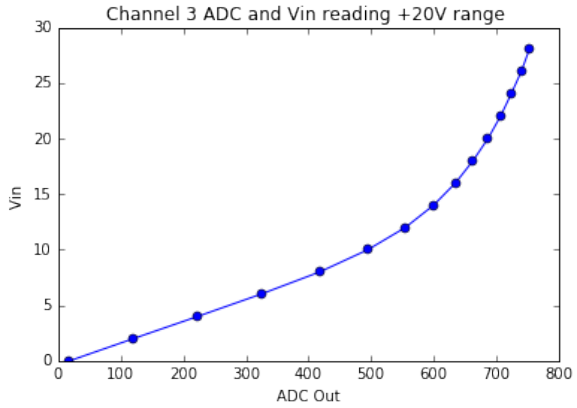
In [210]:
```
print x320
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.  28.]
```

In [211]:
```
print y320
```

```
[  18.  120.  223.  324.  418.  495.  555.  600.  635.  663.  687.
707.
  724.  740.  753.]
```

In [212]:
```
plt.title ('Channel 3 ADC and Vin reading +20V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y320, x320, 'o-')

plt.show()
```



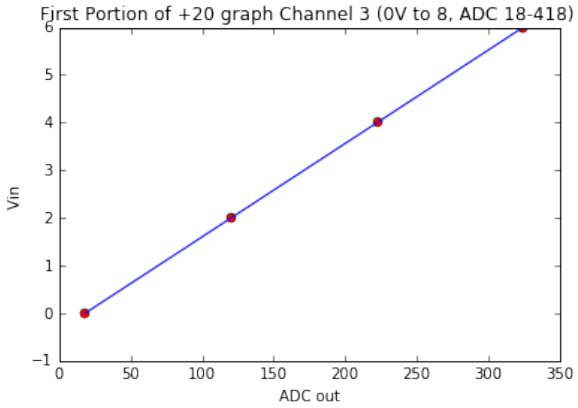
In [213]:
```
z17 = np.polyfit(y320[0:4],x320[0:4],1)

print z17
```

```
[ 0.01958838 -0.35450931]
```

In [214]:
```
pz17 = np.poly1d(z17)
```

In [215]:
```
plt.title ('First Portion of +20 graph Channel 3 (0V to 8, ADC 18-418)
')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320[0:4], x320[0:4],'ro')
plt.plot(y320[0:4], pz17(y320[0:4]), '-')
```

Out[215]: [<matplotlib.lines.Line2D at 0x11359b1d0>]



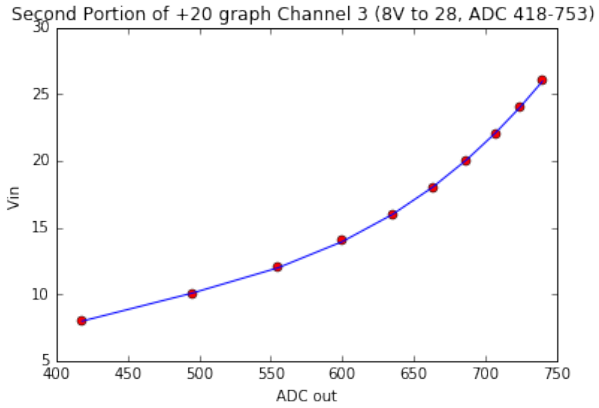
In [216]:
```
z18 = np.polyfit(y320[4:14],x320[4:14],3)

print z18
```

```
[  4.59501798e-07  -6.42990809e-04   3.26363649e-01  -4.96562642e+01
]
```

In [217]:
```
pz18 = np.poly1d(z18)
```

In [218]:
```
plt.title ('Second Portion of +20 graph Channel 3 (8V to 28, ADC 418-753) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320[4:14], x320[4:14],'ro')
plt.plot(y320[4:14], pz18(y320[4:14]), '-')
```

Out[218]:
```
[<matplotlib.lines.Line2D at 0x11420bad0>]
```



In [138]:
```
x360b=np.genfromtxt('Ch3bias60vdata.csv', delimiter=",", usecols=(0))
y360b=np.genfromtxt('Ch3bias60vdata.csv', delimiter=",", usecols=(1))
```

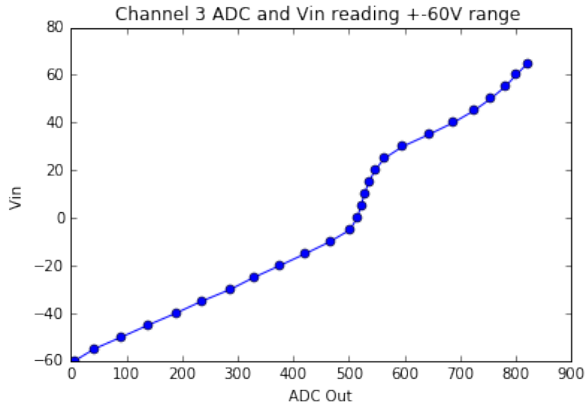
In [139]:
```
print x360b
```

```
[-60. -55. -50. -45. -40. -35. -30. -25. -20. -15. -10.  -5.   0.
5.  10.
  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  65.]
```

In [140]:
```
print y360b
```

```
[   7.   42.   91.  139.  189.  235.  287.  329.  376.  422.  466.
502.
  515.  522.  528.  536.  547.  564.  597.  644.  688.  725.  755.
781.
  801.  822.]
```

In [141]:
```
plt.title ('Channel 3 ADC and Vin reading +-60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y360b, x360b, 'o-')

plt.show()
```



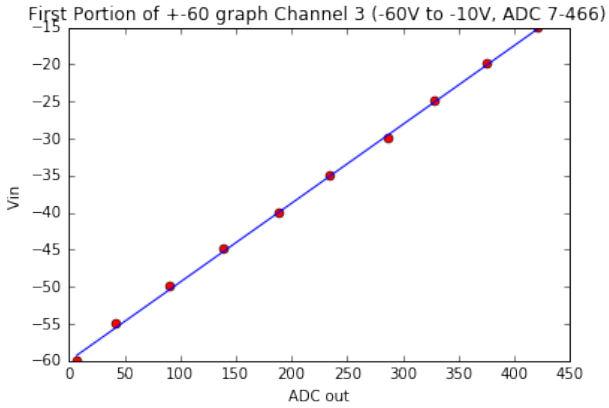
In [197]:
```
z21 = np.polyfit(y360b[0:10],x360b[0:10],1)

print z21
```

```
[  0.10634411 -60.01304714]
```

In [198]:
```
pz21 = np.poly1d(z21)
```

In [200]:
```
plt.title ('First Portion of +-60 graph Channel 3 (-60V to -10V, ADC 7
-466) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360b[0:10], x360b[0:10],'ro')
plt.plot(y360b[0:10], pz21(y360b[0:10]), '-')
```

Out[200]:
```
[<matplotlib.lines.Line2D at 0x111ccf550>]
```



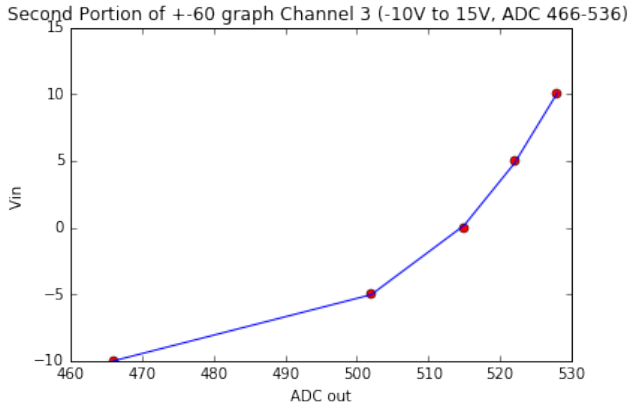
In [203]:
```
z22 = np.polyfit(y360b[10:15],x360b[10:15],3)

print z22
```

```
[  1.38387529e-04  -1.99873792e-01   9.63164326e+01  -1.54937468e+04
]
```

In [204]:
```
pz22 = np.poly1d(z22)
```

In [205]:
```
plt.title ('Second Portion of +-60 graph Channel 3 (-10V to 15V, ADC 4
66-536) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360b[10:15], x360b[10:15],'ro')
plt.plot(y360b[10:15], pz22(y360b[10:15]), '-')
```

Out[205]:
```
[<matplotlib.lines.Line2D at 0x10f703f10>]
```



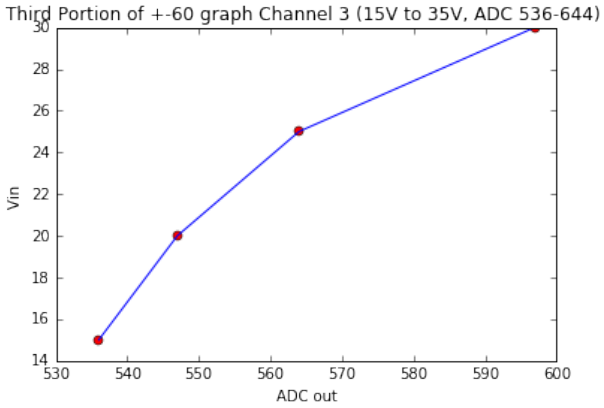
In [158]:
```
z23 = np.polyfit(y360b[15:19],x360b[15:19],3)

print z23
```

```
[  4.71723712e-05  -8.34224599e-02   4.93036761e+01  -9.70893574e+03
]
```

In [159]:
```
pz23 = np.poly1d(z23)
```

In [161]:
```
plt.title ('Third Portion of +-60 graph Channel 3 (15V to 35V, ADC 536
-644) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360b[15:19], x360b[15:19],'ro')
plt.plot(y360b[15:19], pz23(y360b[15:19]), '-')
```

Out[161]:
```
[<matplotlib.lines.Line2D at 0x111a4ea90>]
```



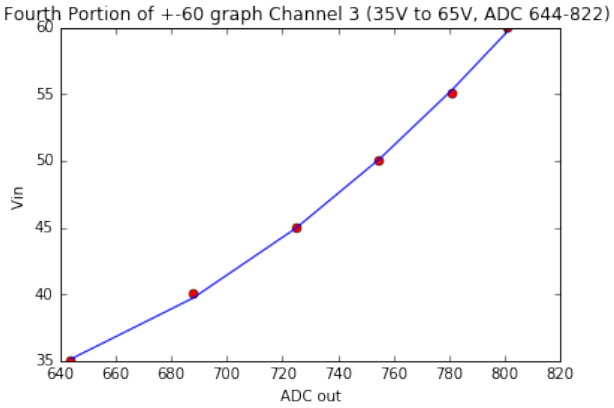
In [162]:
```
z24 = np.polyfit(y360b[19:25],x360b[19:25],2)

print z24
```

```
[  4.63716249e-04  -5.13499415e-01   1.73509826e+02]
```

In [163]:
```
pz24 = np.poly1d(z24)
```

In [164]:
```
plt.title ('Fourth Portion of +-60 graph Channel 3 (35V to 65V, ADC 64
4-822) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360b[19:25], x360b[19:25],'ro')
plt.plot(y360b[19:25], pz24(y360b[19:25]), '-')
```

Out[164]:
```
[<matplotlib.lines.Line2D at 0x10e072d50>]
```



In [169]:
```
x320b=np.genfromtxt('Ch3bias20vdata.csv', delimiter=",", usecols=(0))
y320b=np.genfromtxt('Ch3bias20vdata.csv', delimiter=",", usecols=(1))
```

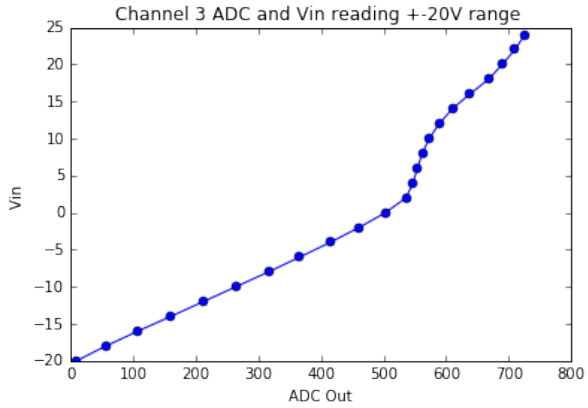
In [170]:
```
print x320b
```

```
[-20. -18. -16. -14. -12. -10.  -8.  -6.  -4.  -2.   0.   2.   4.
6.   8.
  10.  12.  14.  16.  18.  20.  22.  24.]
```

In [171]:
```
print y320b
```

```
[   9.   56.  107.  160.  212.  264.  316.  366.  415.  461.  503.
536.
  547.  554.  562.  573.  589.  610.  638.  668.  690.  710.  726.]
```

In [172]:
```
plt.title ('Channel 3 ADC and Vin reading +-20V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y320b, x320b, 'o-')

plt.show()
```



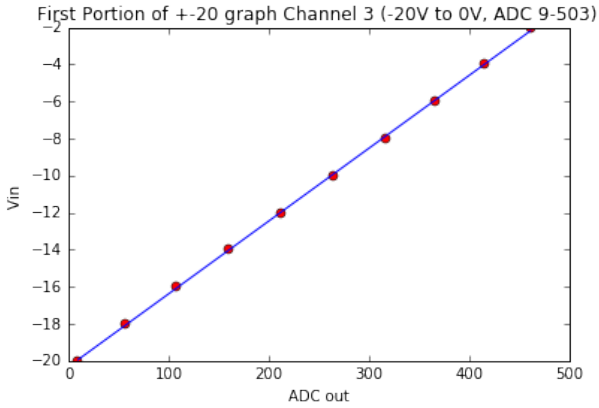
In [183]:
```
z25 = np.polyfit(y320b[0:10],x320b[0:10],1)

print z25
```

```
[  0.03929453 -20.29708651]
```

In [184]:
```
pz25 = np.poly1d(z25)
```

In [185]:
```
plt.title ('First Portion of +-20 graph Channel 3 (-20V to 0V, ADC 9-5
03) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320b[0:10], x320b[0:10],'ro')
plt.plot(y320b[0:10], pz25(y320b[0:10]), '-')
```

Out[185]:
```
[<matplotlib.lines.Line2D at 0x112333b10>]
```



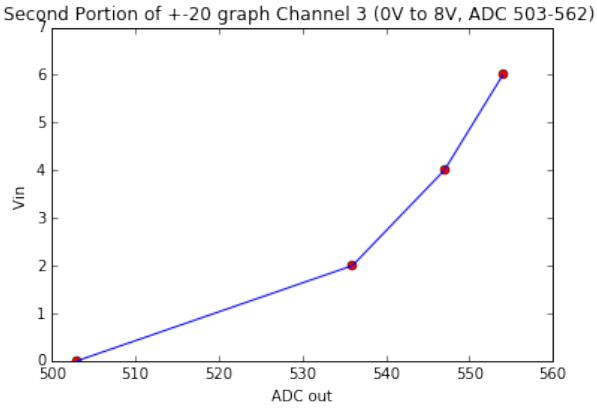
In [188]:
```
z26 = np.polyfit(y320b[10:14],x320b[10:14],3)

print z26
```

```
[  5.91604870e-05  -9.10737114e-02   4.67713447e+01  -8.01248998e+03
]
```

In [189]:
```
pz26 = np.poly1d(z26)
```

In [190]:
```
plt.title ('Second Portion of +-20 graph Channel 3 (0V to 8V, ADC 503-562) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320b[10:14], x320b[10:14],'ro')
plt.plot(y320b[10:14], pz26(y320b[10:14]), '-')
```

Out[190]:
```
[<matplotlib.lines.Line2D at 0x1125fb450>]
```



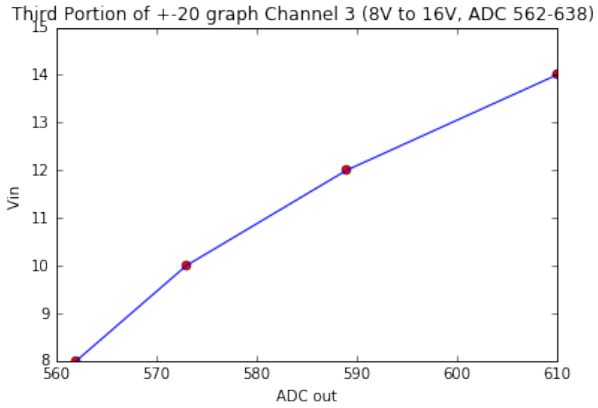
In [191]:
```
z27 = np.polyfit(y320b[14:18],x320b[14:18],3)

print z27
```

```
[  2.70833604e-05  -4.87960905e-02   2.93974651e+01  -5.90883668e+03
]
```

In [192]:
```
pz27 = np.poly1d(z27)
```

In [193]:
```
plt.title ('Third Portion of +-20 graph Channel 3 (8V to 16V, ADC 562-
638) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320b[14:18], x320b[14:18],'ro')
plt.plot(y320b[14:18], pz27(y320b[14:18]), '-')
```

Out[193]: [<matplotlib.lines.Line2D at 0x1127895d0>]



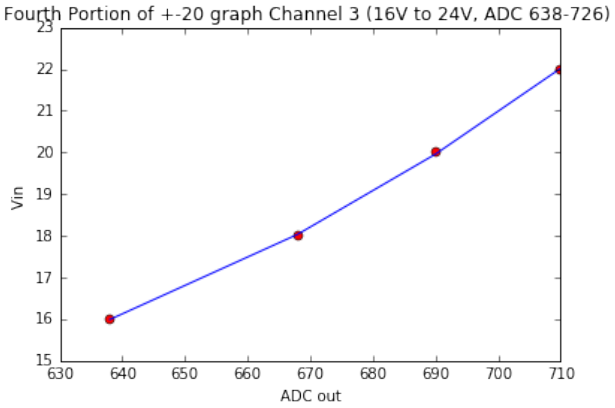
In [194]:
```
z28 = np.polyfit(y320b[18:22],x320b[18:22],2)

print z28
```

```
[  3.70271737e-04  -4.15449140e-01   1.30330998e+02]
```

In [195]:
```
pz28 = np.poly1d(z28)
```

In [196]:
```
plt.title ('Fourth Portion of +-20 graph Channel 3 (16V to 24V, ADC 63
8-726) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320b[18:22], x320b[18:22],'ro')
plt.plot(y320b[18:22], pz28(y320b[18:22]), '-')
```

Out[196]:
```
[<matplotlib.lines.Line2D at 0x113165e50>]
```



In [264]:
```
x360ac=np.genfromtxt('Ch3AC60vdata.csv', delimiter=",", usecols=(0))
y360ac=np.genfromtxt('Ch3AC60vdata.csv', delimiter=",", usecols=(1))
```

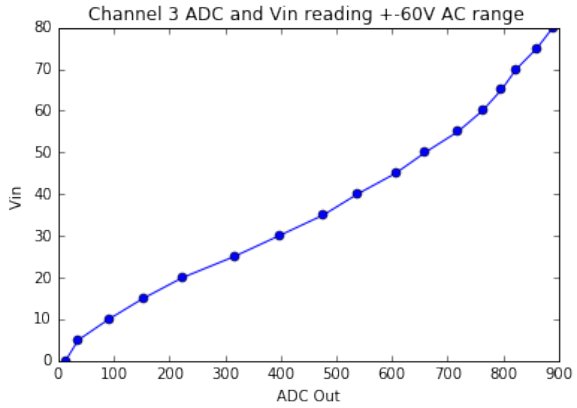
In [265]:
```
print x360ac
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.]
```

In [266]:
```
print y360ac
```

```
[  13.   36.   91.  153.  224.  316.  397.  477.  538.  607.  660.
719.
  763.  797.  824.  861.  890.]
```

In [267]:
```
plt.title ('Channel 3 ADC and Vin reading +-60V AC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y360ac, x360ac, 'o-')

plt.show()
```



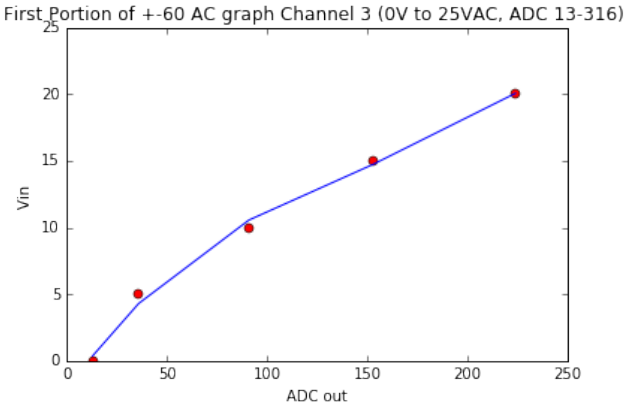
In [268]:
```
z29 = np.polyfit(y360ac[0:5],x360ac[0:5],3)

print z29
```

```
[  2.40912599e-06  -1.07040998e-03   2.18672490e-01  -2.29882854e+00
]
```

In [249]:
```
pz29 = np.poly1d(z29)
```

In [269]:
```
plt.title ('First Portion of +-60 AC graph Channel 3 (0V to 25VAC, ADC
13-316) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360ac[0:5], x360ac[0:5],'ro')
plt.plot(y360ac[0:5], pz29(y360ac[0:5]), '-')
```

Out[269]:
```
[<matplotlib.lines.Line2D at 0x116579a10>]
```



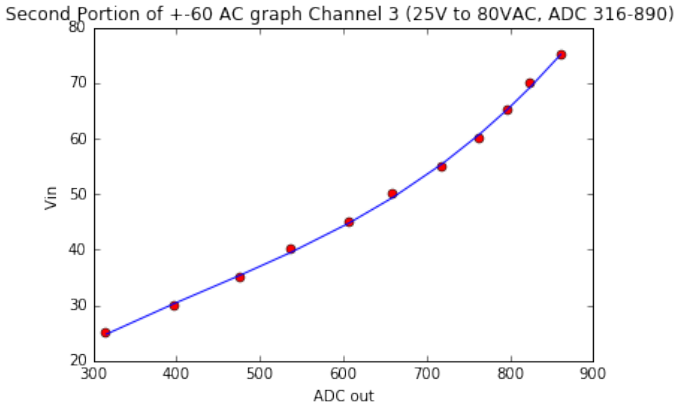
In [270]:
```
z30 = np.polyfit(y360ac[5:16],x360ac[5:16],3)

print z30
```

```
[  1.90678554e-07  -2.47037481e-04   1.70832813e-01  -1.05325295e+01
]
```

In [271]:
```
pz30 = np.poly1d(z30)
```

In [272]:
```
plt.title ('Second Portion of +-60 AC graph Channel 3 (25V to 80VAC, A
DC 316-890) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y360ac[5:16], x360ac[5:16],'ro')
plt.plot(y360ac[5:16], pz30(y360ac[5:16]), '-')
```

Out[272]:
```
[<matplotlib.lines.Line2D at 0x1166fd3d0>]
```



In [259]:
```
x320ac=np.genfromtxt('Ch3AC20vdata.csv', delimiter=",", usecols=(0))
y320ac=np.genfromtxt('Ch3AC20vdata.csv', delimiter=",", usecols=(1))
```

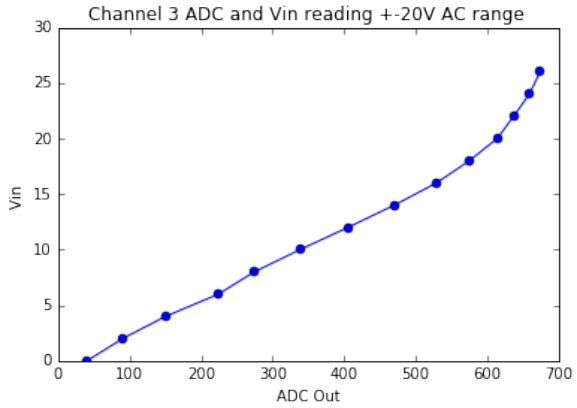
In [260]:
```
print x320ac
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.]
```

In [261]:
```
print y320ac
```

```
[  40.   90.  150.  224.  274.  338.  405.  470.  529.  575.  614.
637.
  659.  674.]
```

In [263]:
```
plt.title ('Channel 3 ADC and Vin reading +-20V AC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y320ac, x320ac, 'o-')

plt.show()
```



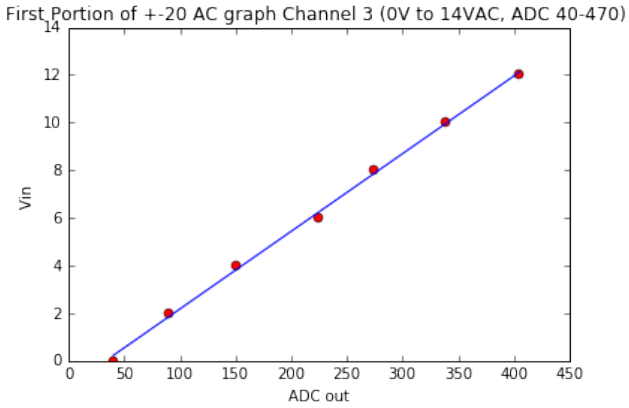
In [273]:
```
z31 = np.polyfit(y320ac[0:7],x320ac[0:7],1)

print z31
```

```
[ 0.03259544 -1.08252444]
```

In [275]:
```
pz31 = np.poly1d(z31)
```

In [276]:
```
plt.title ('First Portion of +-20 AC graph Channel 3 (0V to 14VAC, ADC
40-470) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320ac[0:7], x320ac[0:7],'ro')
plt.plot(y320ac[0:7], pz31(y320ac[0:7]), '-')
```

Out[276]:
```
[<matplotlib.lines.Line2D at 0x116b51d50>]
```



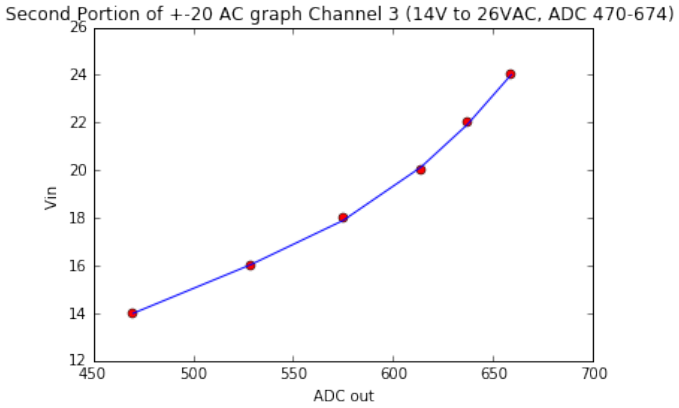
In [280]:
```
z32 = np.polyfit(y320ac[7:13],x320ac[7:13],3)

print z32
```

```
[  1.05180291e-06  -1.60346156e-03   8.48433881e-01  -1.39767537e+02
]
```

In [281]:
```
pz32 = np.poly1d(z32)
```

In [282]:
```
plt.title ('Second Portion of +-20 AC graph Channel 3 (14V to 26VAC, A
DC 470-674) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y320ac[7:13], x320ac[7:13],'ro')
plt.plot(y320ac[7:13], pz32(y320ac[7:13]), '-')
```

Out[282]:
```
[<matplotlib.lines.Line2D at 0x116e30f10>]
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [402]:
```
x460=np.genfromtxt('Ch460vdata.csv', delimiter=",", usecols=(0))
y460=np.genfromtxt('Ch460vdata.csv', delimiter=",", usecols=(1))
```

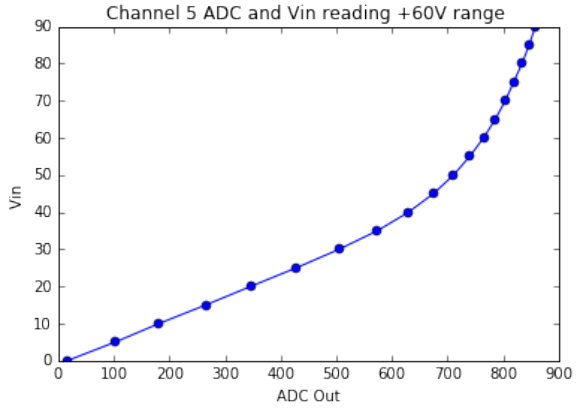
In [403]:
```
print x460
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.  85.  90.]
```

In [404]:
```
print y460
```

```
[  16.  102.  181.  265.  346.  428.  505.  574.  630.  675.  711.
740.
  765.  786.  804.  819.  834.  847.  858.]
```

In [405]:
```
plt.title ('Channel 5 ADC and Vin reading +60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y460, x460, 'o-')

plt.show()
```



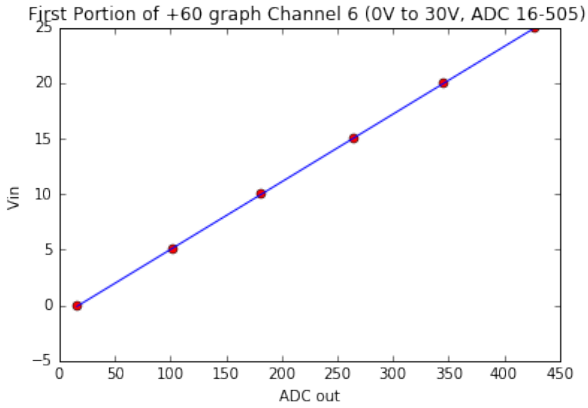
In [406]:
```
z19 = np.polyfit(y460[0:6],x460[0:6],1)

print z19
```

```
[ 0.06084352 -1.06810412]
```

In [407]:
```
pz19 = np.poly1d(z19)
```

In [408]:
```
plt.title ('First Portion of +60 graph Channel 6 (0V to 30V, ADC 16-50
5) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460[0:6], x460[0:6],'ro')
plt.plot(y460[0:6], pz19(y460[0:6]), '-')
```

Out[408]: [<matplotlib.lines.Line2D at 0x11ac49910>]



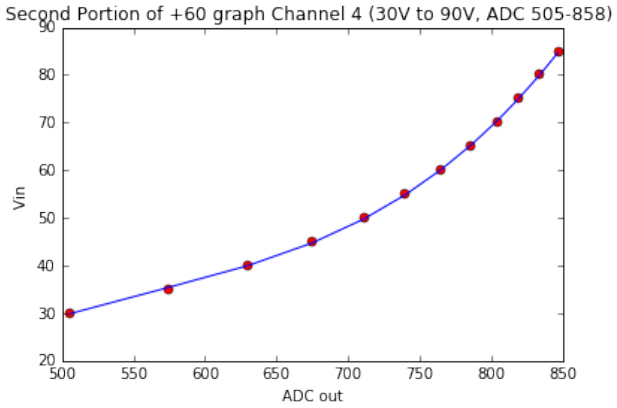
In [409]:
```
z20 = np.polyfit(y460[6:18],x460[6:18],3)

print z20
```

```
[  1.27468593e-06  -2.15988023e-03   1.29593845e+00  -2.37930937e+02
]
```

In [410]:
```
pz20 = np.poly1d(z20)
```

In [411]:
```
plt.title ('Second Portion of +60 graph Channel 4 (30V to 90V, ADC 505
-858) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460[6:18], x460[6:18],'ro')
plt.plot(y460[6:18], pz20(y460[6:18]), '-')
```

Out[411]:
```
[<matplotlib.lines.Line2D at 0x11941ba10>]
```



In [412]:
```
x420=np.genfromtxt('Ch420vdata.csv', delimiter=",", usecols=(0))
y420=np.genfromtxt('Ch420vdata.csv', delimiter=",", usecols=(1))
```

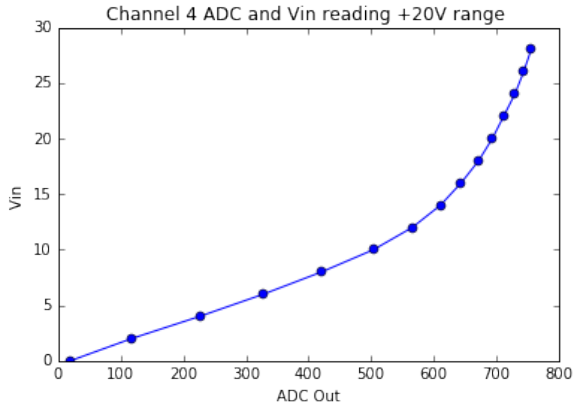
In [413]:
```
print x420
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.  28.]
```

In [414]:
```
print y420
```

```
[  19.  117.  226.  328.  421.  504.  566.  611.  644.  672.  694.
712.
  730.  744.  756.]
```

In [415]:
```
plt.title ('Channel 4 ADC and Vin reading +20V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y420, x420, 'o-')

plt.show()
```



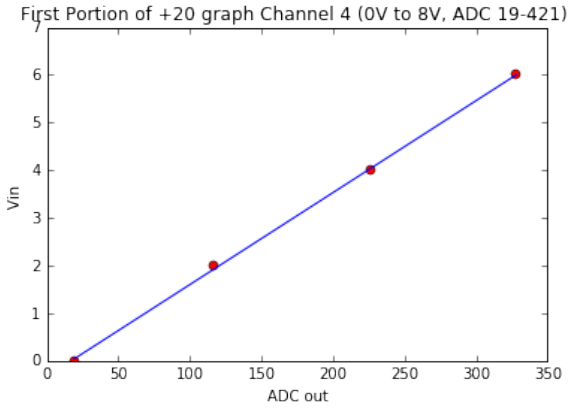
In [416]:
```
z33 = np.polyfit(y420[0:4],x420[0:4],1)

print z33
```

```
[ 0.01929776 -0.32886281]
```

In [417]:
```
pz33 = np.poly1d(z33)
```

In [418]:
```
plt.title ('First Portion of +20 graph Channel 4 (0V to 8V, ADC 19-421
) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420[0:4], x420[0:4],'ro')
plt.plot(y420[0:4], pz33(y420[0:4]), '-')
```

Out[418]:
```
[<matplotlib.lines.Line2D at 0x11b2d1450>]
```



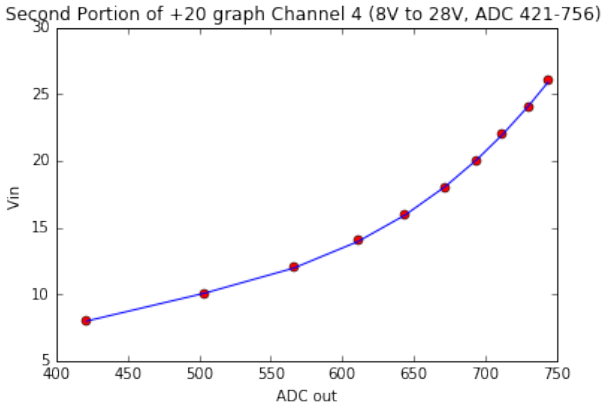
In [419]:
```
z34 = np.polyfit(y420[4:14],x420[4:14],3)

print z34
```

```
[  5.18158013e-07  -7.39041249e-04   3.75546201e-01  -5.78020384e+01
]
```

In [420]:
```
pz34 = np.poly1d(z34)
```

In [421]:
```
plt.title ('Second Portion of +20 graph Channel 4 (8V to 28V, ADC 421-
756) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420[4:14], x420[4:14],'ro')
plt.plot(y420[4:14], pz34(y420[4:14]), '-')
```

Out[421]:
```
[<matplotlib.lines.Line2D at 0x11b59ad50>]
```



In [3]:
```
x460b=np.genfromtxt('Ch4bias60vdata.csv', delimiter=",", usecols=(0))
y460b=np.genfromtxt('Ch4bias60vdata.csv', delimiter=",", usecols=(1))
```

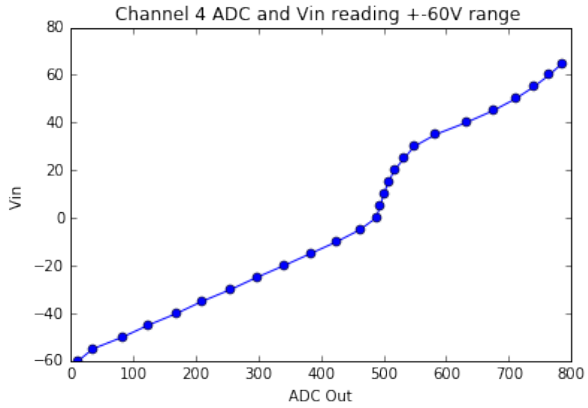
In [4]:
```
print x460b
```

```
[-60. -55. -50. -45. -40. -35. -30. -25. -20. -15. -10.  -5.   0.
5.  10.
  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  65.]
```

In [5]:
```
print y460b
```

```
[  12.   35.   83.  124.  170.  210.  256.  297.  341.  383.  425.
462.
  488.  495.  500.  508.  517.  533.  549.  583.  632.  675.  711.
740.
  765.  786.]
```

In [6]:
```
plt.title ('Channel 4 ADC and Vin reading +-60V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y460b, x460b, 'o-')

plt.show()
```



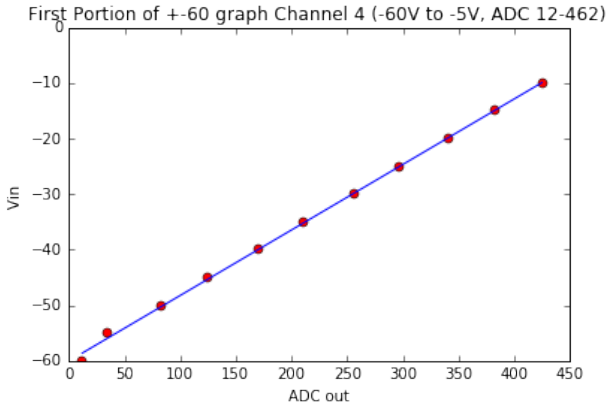
In [7]:
```
z35 = np.polyfit(y460b[0:11],x460b[0:11],1)

print z35
```

```
[  0.11781646 -60.01993276]
```

In [8]:
```
pz35 = np.poly1d(z35)
```

In [9]:
```
plt.title ('First Portion of +-60 graph Channel 4 (-60V to -5V, ADC 12
-462) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460b[0:11], x460b[0:11],'ro')
plt.plot(y460b[0:11], pz35(y460b[0:11]), '-')
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x10c84b610>]
```



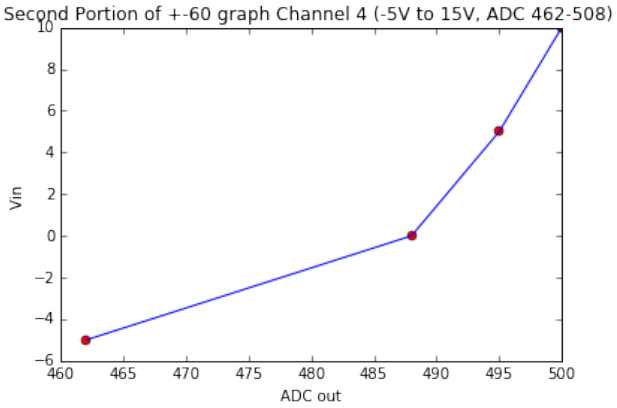
In [10]:
```
z36 = np.polyfit(y460b[11:15],x460b[11:15],3)

print z36
```

```
[  2.10316000e-04  -2.88089104e-01   1.31483771e+02  -1.99991093e+04
]
```

In [11]:
```
pz36 = np.poly1d(z36)
```

In [12]:
```
plt.title ('Second Portion of +-60 graph Channel 4 (-5V to 15V, ADC 46
2-508) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460b[11:15], x460b[11:15],'ro')
plt.plot(y460b[11:15], pz36(y460b[11:15]), '-')
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x10c953bd0>]
```



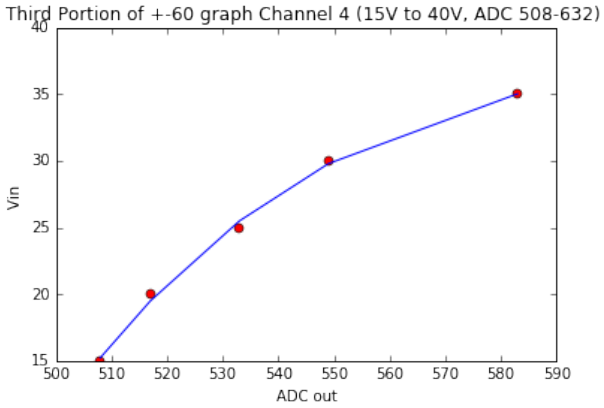
In [13]:
```
z37 = np.polyfit(y460b[15:20],x460b[15:20],3)

print z37
```

```
[  1.56085684e-05  -2.82788058e-02   1.71602086e+01  -3.45064843e+03
]
```

In [14]:
```
pz37 = np.poly1d(z37)
```

In [15]:
```
plt.title ('Third Portion of +-60 graph Channel 4 (15V to 40V, ADC 508
-632) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460b[15:20], x460b[15:20],'ro')
plt.plot(y460b[15:20], pz37(y460b[15:20]), '-')
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x10cb2a510>]
```



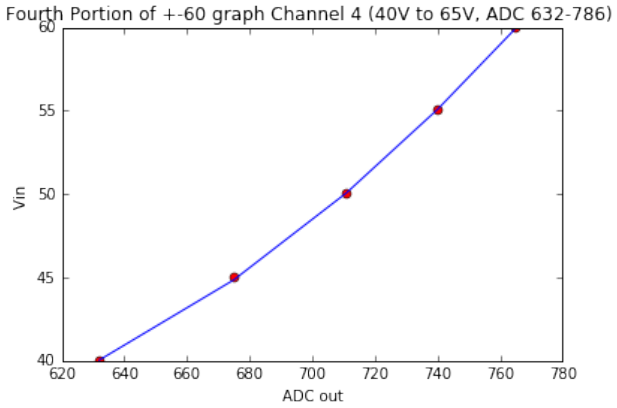
In [16]:
```
z38 = np.polyfit(y460b[20:25],x460b[20:25],2)

print z38
```

```
[  4.17850145e-04  -4.34178446e-01   1.47549041e+02]
```

In [17]:
```
pz38 = np.poly1d(z38)
```

In [18]:
```
plt.title ('Fourth Portion of +-60 graph Channel 4 (40V to 65V, ADC 63
2-786) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460b[20:25], x460b[20:25],'ro')
plt.plot(y460b[20:25], pz38(y460b[20:25]), '-')
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x10cc92d50>]
```



In [19]:
```
x420b=np.genfromtxt('Ch4bias20vdata.csv', delimiter=",", usecols=(0))
y420b=np.genfromtxt('Ch4bias20vdata.csv', delimiter=",", usecols=(1))
```

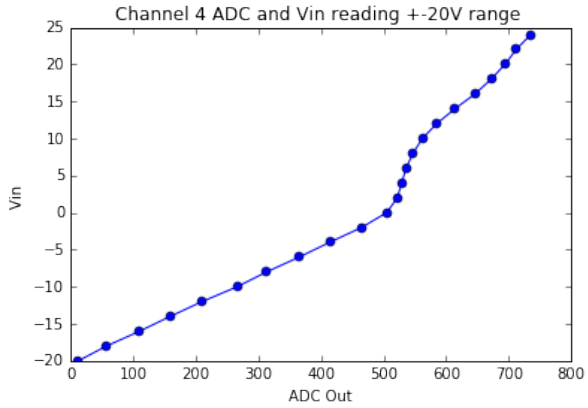
In [20]:
```
print x420b
```

```
[-20. -18. -16. -14. -12. -10.  -8.  -6.  -4.  -2.   0.   2.   4.
6.   8.
  10.  12.  14.  16.  18.  20.  22.  24.]
```

In [21]:
```
print y420b
```

```
[  12.   57.  110.  159.  210.  266.  313.  365.  414.  465.  505.
522.
  529.  536.  547.  562.  585.  614.  646.  672.  694.  712.  735.]
```

In [22]:
```
plt.title ('Channel 4 ADC and Vin reading +-20V range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y420b, x420b, 'o-')

plt.show()
```



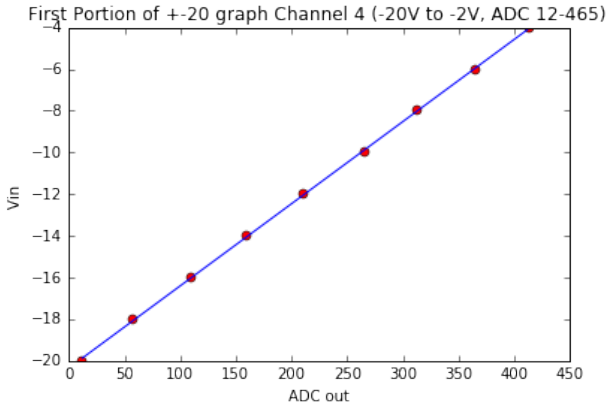
In [23]:
```
z39 = np.polyfit(y420b[0:9],x420b[0:9],1)

print z39
```

```
[  0.03939923 -20.34388101]
```

In [24]:
```
pz39 = np.poly1d(z39)
```

In [25]:
```
plt.title ('First Portion of +-20 graph Channel 4 (-20V to -2V, ADC 12
-465) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420b[0:9], x420b[0:9],'ro')
plt.plot(y420b[0:9], pz39(y420b[0:9]), '-')
```

Out[25]:
```
[<matplotlib.lines.Line2D at 0x10cf5de90>]
```



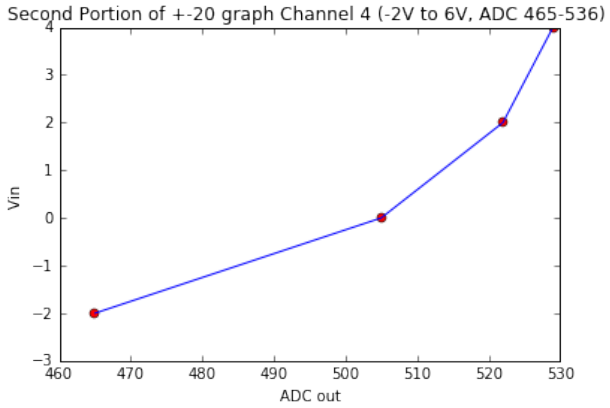
In [26]:
```
z40 = np.polyfit(y420b[9:13],x420b[9:13],3)

print z40
```

```
[  9.08751659e-05  -1.34398957e-01   6.62523055e+01  -1.08859171e+04
]
```

In [27]:
```
pz40 = np.poly1d(z40)
```

In [28]:
```
plt.title ('Second Portion of +-20 graph Channel 4 (-2V to 6V, ADC 465
-536) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420b[9:13], x420b[9:13],'ro')
plt.plot(y420b[9:13], pz40(y420b[9:13]), '-')
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x10c46b890>]
```



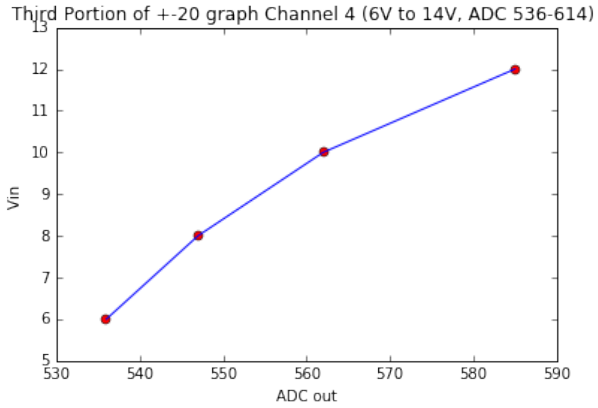
In [29]:
```
z41 = np.polyfit(y420b[13:17],x420b[13:17],3)

print z41
```

```
[  1.31501929e-05  -2.34968693e-02   1.40607423e+01  -2.80500816e+03
]
```

In [30]:
```
pz41 = np.poly1d(z41)
```

In [36]:
```
plt.title ('Third Portion of +-20 graph Channel 4 (6V to 14V, ADC 536-614) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420b[13:17], x420b[13:17],'ro')
plt.plot(y420b[13:17], pz41(y420b[13:17]), '-')
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x10d430950>]
```



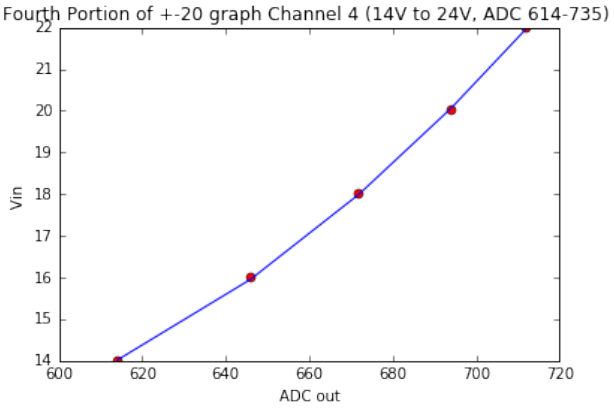
In [37]:
```
z42 = np.polyfit(y420b[17:22],x420b[17:22],2)

print z42
```

```
[  3.09781869e-04  -3.29670592e-01   9.96480572e+01]
```

In [38]:
```
pz42 = np.poly1d(z42)
```

In [39]:
```
plt.title ('Fourth Portion of +-20 graph Channel 4 (14V to 24V, ADC 61
4-735) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420b[17:22], x420b[17:22],'ro')
plt.plot(y420b[17:22], pz42(y420b[17:22]), '-')
```

Out[39]:
```
[<matplotlib.lines.Line2D at 0x10d878290>]
```



In [371]:
```
x460ac=np.genfromtxt('Ch4AC60vdata.csv', delimiter=",", usecols=(0))
y460ac=np.genfromtxt('Ch4AC60vdata.csv', delimiter=",", usecols=(1))
```

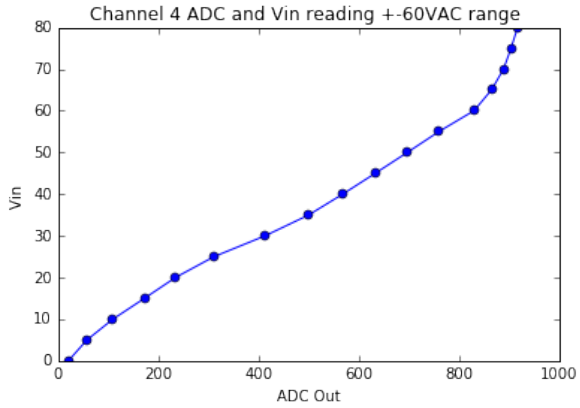
In [372]:
```
print x460ac
```

```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  6
5.  70.
  75.  80.]
```

In [373]:
```
print y460ac
```

```
[  20.   57.  109.  172.  234.  311.  413.  500.  568.  633.  697.
760.
  830.  865.  889.  904.  917.]
```

In [374]:
```
plt.title ('Channel 4 ADC and Vin reading +-60VAC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y460ac, x460ac, 'o-')

plt.show()
```



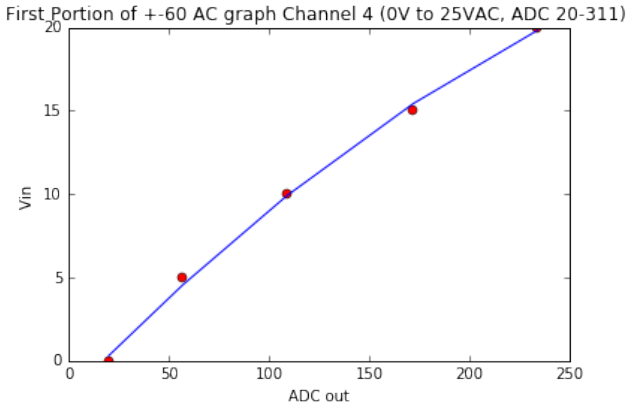
In [382]:
```
z43 = np.polyfit(y460ac[0:5],x460ac[0:5],2)

print z43
```

```
[ -1.33665982e-04   1.25135292e-01  -2.14617421e+00]
```

In [383]:
```
pz43 = np.poly1d(z43)
```

In [384]:
```
plt.title ('First Portion of +-60 AC graph Channel 4 (0V to 25VAC, ADC
20-311) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460ac[0:5], x460ac[0:5],'ro')
plt.plot(y460ac[0:5], pz43(y460ac[0:5]), '-')
```

Out[384]:
```
[<matplotlib.lines.Line2D at 0x11a4c0110>]
```



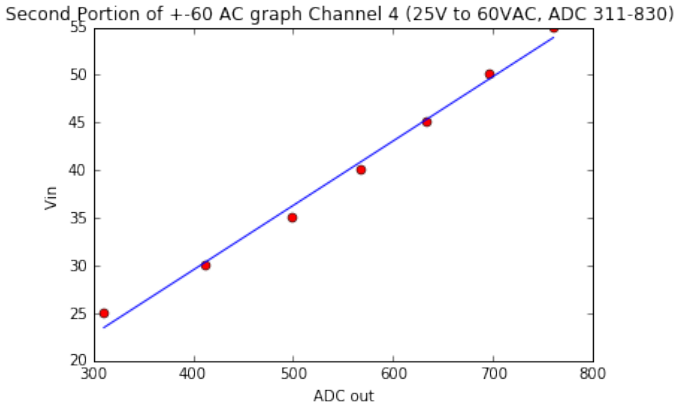
In [385]:
```
z44 = np.polyfit(y460ac[5:12],x460ac[5:12],1)

print z44
```

```
[ 0.06774023  2.43320481]
```

In [386]:
```
pz44 = np.poly1d(z44)
```

In [387]:
```
plt.title ('Second Portion of +-60 AC graph Channel 4 (25V to 60VAC, A
DC 311-830) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460ac[5:12], x460ac[5:12],'ro')
plt.plot(y460ac[5:12], pz44(y460ac[5:12]), '-')
```

Out[387]:
```
[<matplotlib.lines.Line2D at 0x11a934210>]
```



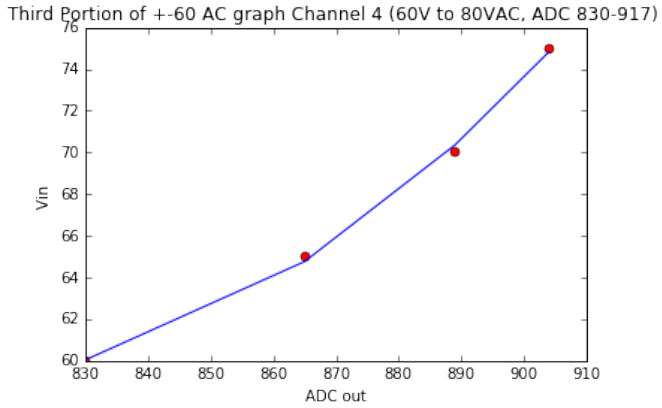
In [388]:
```
z45 = np.polyfit(y460ac[12:16],x460ac[12:16],2)

print z45
```

```
[  1.66286043e-03  -2.68374256e+00   1.14201152e+03]
```

In [389]:
```
pz45 = np.poly1d(z45)
```

In [390]:
```
plt.title ('Third Portion of +-60 AC graph Channel 4 (60V to 80VAC, AD
C 830-917) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y460ac[12:16], x460ac[12:16],'ro')
plt.plot(y460ac[12:16], pz45(y460ac[12:16]), '-')
```

Out[390]:
```
[<matplotlib.lines.Line2D at 0x11aa72c90>]
```



In [391]:
```
x420ac=np.genfromtxt('Ch4AC20vdata.csv', delimiter=",", usecols=(0))
y420ac=np.genfromtxt('Ch4AC20vdata.csv', delimiter=",", usecols=(1))
```

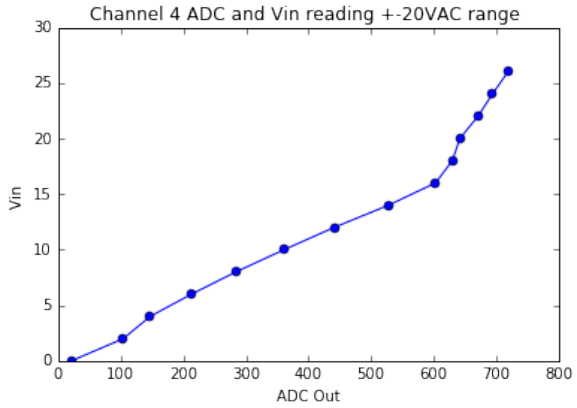
In [392]:
```
print x420ac
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  2
6.]
```

In [393]:
```
print y420ac
```

```
[  21.  104.  147.  213.  284.  361.  441.  528.  603.  630.  642.
671.
  694.  719.]
```

In [394]:
```
plt.title ('Channel 4 ADC and Vin reading +-20VAC range ')
plt.xlabel('ADC Out')
plt.ylabel('Vin ')

plt.plot(y420ac, x420ac, 'o-')

plt.show()
```



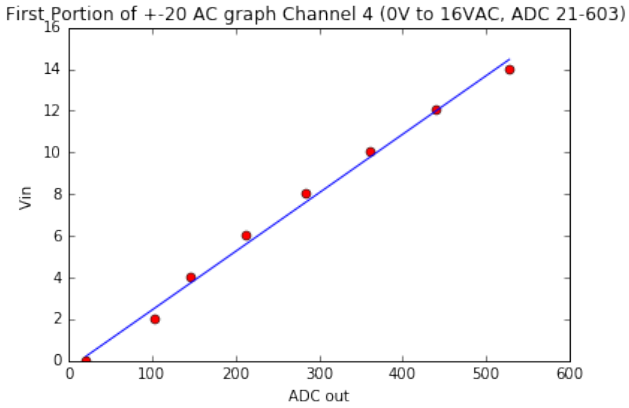
In [396]:
```
z46 = np.polyfit(y420ac[0:8],x420ac[0:8],1)

print z46
```

```
[ 0.02808476 -0.36873818]
```

In [397]:
```
pz46 = np.poly1d(z46)
```

In [398]:
```
plt.title ('First Portion of +-20 AC graph Channel 4 (0V to 16VAC, ADC
21-603) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420ac[0:8], x420ac[0:8],'ro')
plt.plot(y420ac[0:8], pz46(y420ac[0:8]), '-')
```

Out[398]:
```
[<matplotlib.lines.Line2D at 0x119d48e10>]
```



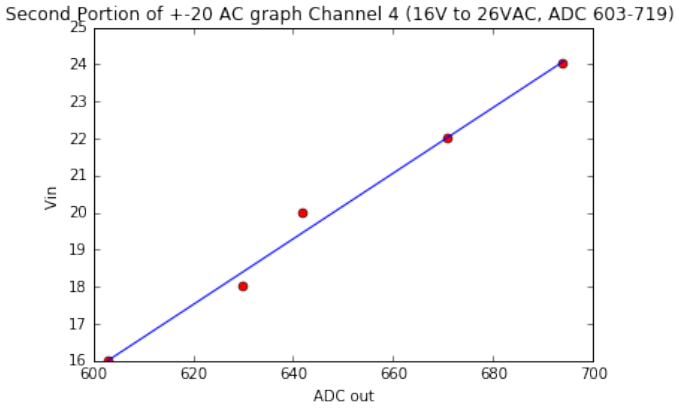
In [399]:
```
z47 = np.polyfit(y420ac[8:13],x420ac[8:13],1)

print z47
```

```
[  0.08866799 -37.45685885]
```

In [400]:
```
pz47 = np.poly1d(z47)
```

In [401]:
```
plt.title ('Second Portion of +-20 AC graph Channel 4 (16V to 26VAC, A
DC 603-719) ')
plt.xlabel('ADC out')
plt.ylabel('Vin ')

plt.plot(y420ac[8:13], x420ac[8:13],'ro')
plt.plot(y420ac[8:13], pz47(y420ac[8:13]), '-')
```

Out[401]:
```
[<matplotlib.lines.Line2D at 0x118e8b150>]
```



In [ ]:

Appendix C – C Source Code

```
//*****************************************************************************
// Operates Stellaris LM3S8962 board for controlling shield circuit,
functioning
// as a 4 channel AC/DC voltmeter.
// Composed by Callie Muckelroy for Master's Report for University of Texas at
Austin,
// MSE
// email: muckelroy@utexas.edu
// Copyright 2016, University of Texas at Austin,
// Supervising Professors: Bill Bard and Jonathan  Valvano
// Supporting header files are copyright Texas Instruments
//*****************************************************************************

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "rit128x96x4.h"
#include "driverlib/pwm.h"
#include "utils/uartstdio.h"
#include "driverlib/uart.h"
#include <math.h>
#include <limits.h>
#include <stdio.h>

// range is a flag for the voltage range being measured. Each
// channel begins at range = 1.
int range_1=1;
int range_2=1;
int range_3=1;
int range_4=1;

// used as counter for
// adjusting the refresh rate of the display.
// Initializes at zero. Refresh rate is set by
// modifying max count in while() loop.
int displayRate = 0;

// Used as FIFOs to average the measurement over the course of
// 80 cycle counts.
double ADCValueAvg1[80];
double ADCValueAvg2[80];
double ADCValueAvg3[80];
double ADCValueAvg4[80];

```

```
// Used for mean FIFO calculation
long Sum1=0;
long Sum2=0;
long Sum3=0;
long Sum4=0;

// Store the maximum and minimum of each of the above FIFOs
// for all 4 channels.
double maximum1;
double minimum1;
double maximum2;
double minimum2;
double maximum3;
double minimum3;
double maximum4;
double minimum4;

// The mean value stored in FIFO is calculated and
// stored in each of these 4 variables.
long ch1ADCDisplay;
long ch2ADCDisplay;
long ch3ADCDisplay;
long ch4ADCDisplay;

// counters used for DCAvg function FIFOs
int i1=0;
int i2=0;
int i3=0;
int i4=0;

// counters used for max/min calculation counters as they cycles through the
ADCValueAvg1 buffers
int imax1 = 0;
int imax2 = 0;
int imax3 = 0;
int imax4 = 0;

// used as counters determine if/how to switch out of range 1 for channel 1.
int i1_1 = 0;
int i1_c1 = 0;
int i1_c2 = 0;
int i1_2 = 0;
int i1_c3 = 0;
int i1_c4 = 0;
int i1_3 = 0;
int i1_c5 = 0;
int i1_c6 = 0;
int i1_4 = 0;
```

```
int i1_c7 = 0;
int i1_c8 = 0;
int i1_5 = 0;
int i1_c9;
int i1_6;
int i1_c10;

// used as counters determine if/how to switch out of range 1 for channel 2.
int i2_1 = 0;
int i2_c1 = 0;
int i2_c2 = 0;
int i2_2 = 0;
int i2_c3 = 0;
int i2_c4 = 0;
int i2_3 = 0;
int i2_c5 = 0;
int i2_c6 = 0;
int i2_4 = 0;
int i2_c7 = 0;
int i2_c8 = 0;
int i2_5 = 0;
int i2_c9;
int i2_6;
int i2_c10;

// used as counters determine if/how to switch out of range 1 for channel 2.
int i3_1 = 0;
int i3_c1 = 0;
int i3_c2 = 0;
int i3_2 = 0;
int i3_c3 = 0;
int i3_c4 = 0;
int i3_3 = 0;
int i3_c5 = 0;
int i3_c6 = 0;
int i3_4 = 0;
int i3_c7 = 0;
int i3_c8 = 0;
int i3_5 = 0;
int i3_c9;
int i3_6;
int i3_c10;

// used as counters determine if/how to switch out of ranges for channel 4.
int i4_1 = 0;
int i4_c1 = 0;
int i4_c2 = 0;
int i4_2 = 0;
```

```
int i4_c3 = 0;
int i4_c4 = 0;
int i4_3 = 0;
int i4_c5 = 0;
int i4_c6 = 0;
int i4_4 = 0;
int i4_c7 = 0;
int i4_c8 = 0;
int i4_5 = 0;
int i4_c9;
int i4_6;
int i4_c10;


// the adj_n function transforms the raw ADC value into an actual voltage value.
// Separate functions are in place for each channel, and each range
// within each channel.
unsigned long adj_n(unsigned long in, int range, int channel)
{
    // The ADC value is assigned to "n"
    long n=in;

    if (channel==1)
    {
        // +60V range
        if (range==1)
        {
            if(in > 940)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 537)
            {
                // clear "overload" display
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n = 100*(.000000898034069*pow(in,3)-.00164569796*pow(in,2)+1.0775934*in-213.67343);
            }
            else
            {
                n = 100*(0.05667105*in -0.66607681);
            }
        }

        // +20 V Range
        if (range==2)
```

```
        {
            if(in > 753)
            {
                // Display "OVERLOAD!!"
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 534)
            {
                // clear "overload" display and turn off overload alarm if on.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n =
100*(.000000447800473*pow(in,3)-.000716941086*pow(in,2)+.408798967*in-72.
0598349);
            }
            else
            {
                n = 100*(.01862997*in - .24377945);
            }
        }

        // +- 60V range
        if (range==3)
        {

            if(in > 829)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 688)
            {
                // clear "overload" display.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n = 100*(.000323432752*pow(in,2)-.356694514*in+133.409998);
            }
            else if(in >= 536)
            {
                n =
100*(.0000166780325*pow(in,3)-.0313463796*pow(in,2)+19.7126073*n-4112.19508);
            }
            else if(in >= 499)
            {
                n =
100*(-.000438444002*pow(in,3)+.691726325*pow(in,2)-363.005678*n+63373.913);
            }
            else if(in >= 12)
            {
                n = 100*(.10863971*n-58.35465865);
```

```
            }
            else
            {
                n = 100*(.46153846*n-61.15384615);
            }
        }

        // +- 20V range
        if (range==4)
        {
            // Greater than 764 ADC value is overload.
            if(in > 764)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 703)
            {
                // clear "overload" display.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n = 100*(.000290726437*pow(in,2)-.355365552*in+114.931164);
            }
            else if(in >= 611)
            {
                n =
100*(.0000115042808*pow(in,3)-.0229645386*pow(in,2)+15.3398426*n-3414.56785);
            }
            else if(in >= 556)
            {
                n =
100*(.0000349257851*pow(in,3)-.0576113098*pow(in,2)+31.6927686*n-5812.91422);
            }
            else if(in >= 52)
            {
                n = 100*(.03595532*n-18.94238609);
            }
            else
            {
                n =
100*(.000661743137*pow(in,3)-.0302298547*pow(in,2)+.450641886*n-20.7114737);
            }
        }

        if (range==5)
        {
        // this is used for AC measurement while the bias voltage is applied.
It should be able to measure
        // 60 VAC (with no input DC bias).
```

```
276             if(in > 1015)
277             {
278                 // Display "OVERLOAD!!".
279                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
280             }
281             else if (in >= 490)
282             {
283                 // clear "overload" display.
284                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
285                 n = 100*(.09408478*in-10.94296859);
286             }
287             else
288             {
289                 n = 100*(-.0000753280621*pow(in,2)+.107064736*in-.928507402);
290             }
291         }
292 
293         //20V AC range
294         if (range==6)
295         {
296         // this is used for AC measurement while the bias voltage is applied.
    … It should be able to measure
297         // 20 VAC (with no input DC bias).
298 
299 
300             if(in > 660)
301             {
302                 // Display "OVERLOAD!!".
303                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
304             }
305             else if (in >= 336)
306             {
307                 // clear "overload" display.
308                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
309                 n = 100*(.0000671252055*pow(in,2)-.0235743674*in+10.2917684);
310             }
311             else
312             {
313                 n = 100*(0.03287524*in-.87883366);
314             }
315         }
316     }
317 
318     if (channel==2)
319     {
320         // +60V range
321         if (range==1)
322         {
```

```
            if(in > 885)
                    {
                        // Display "OVERLOAD!!".
                        RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
                    }
            else if(in >= 499)
            {
                // clear "overload" display
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n =
100*(.00000121032951*pow(in,3)-.00206514637*pow(in,2)+1.23923536*in-229.692002)
;
            }
            else
            {
                n = 100*(0.05086454*in -0.75276417);
            }
        }

        // +20 V Range
        if (range==2)
        {
            if(in > 766)
            {
                // Display "OVERLOAD!!"
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 429)
            {
                // clear "overload" display and turn off overload alarm if on.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n =
100*(.000000564814913*pow(in,3)-.000833633285*pow(in,2)+.434261015*in-69.
5018065);
            }
            else
            {
                n = 100*(.01910209*in - .29033538);
            }
        }

        // +- 60V range
        if (range==3)
        {
            if(in > 807)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
```

```
367             }
368             else if(in >= 707)
369             {
370                 // clear "overload" display.
371                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
372                 n = 100*(.000609276138*pow(in,2)-.746959375*in+263.325274);
373             }
374             else if(in >= 530)
375             {
376                 n =
… 100*(.0000303090955*pow(in,3)-.0577381309*pow(in,2)+36.7667665*n-7794.21689);
377             }
378             else
379             {
380                 n = 100*(.10716274*n-59.19245342);
381             }
382         }
383 
384         // +- 20V range
385         if (range==4)
386         {
387             if(in > 746)
388             {
389                 // Display "OVERLOAD!!".
390                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
391             }
392             else if(in >= 663)
393             {
394                 // clear "overload" display.
395                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
396                 n = 100*(.000505050505*pow(in,2)-.627272727*in+209.616162);
397             }
398             else if(in >= 513)
399             {
400                 n =
… 100*(.00000448749242*pow(in,3)-.00941783264*pow(in,2)+6.5822116*n-1517.55640);
401             }
402             else
403             {
404                 n =  100*(.03849697*n-20.16612851);
405             }
406         }
407 
408         if (range==5)
409         {
410         // this is used for AC measurement while the bias voltage is applied.
… It should be able to measure
411         // 60 VAC (with no input DC bias).
```

```
412             if(in > 800)
413             {
414                 // Display "OVERLOAD!!".
415                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
416             }
417             else if (in >= 490)
418             {
419                 // clear "overload" display.
420                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
421                 n = 100*(.09408488*in-10.94296869);
422             }
423             else
424             {
425                 n = 100*(-.0000753880621*pow(in,2)+.107066736*in-.928507502);
426             }
427         }
428 
429         //20V AC range
430         if (range==6)
431         {
432         // this is used for AC measurement while the bias voltage is applied.
    … It should be able to measure
433         // 20 VAC (with no input DC bias).
434             if(in > 660)
435             {
436                 // Display "OVERLOAD!!".
437                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
438             }
439             else if (in >= 336)
440             {
441                 // clear "overload" display.
442                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
443                 n = 100*(.0000671253055*pow(in,2)-.0235743774*in+10.2917784);
444             }
445             else
446             {
447                 n = 100*(0.03287624*in-.87883376);
448             }
449         }
450     }
451 
452     if (channel==3)
453     {
454         // +60V range
455         if (range==1)
456         {
457             if(in > 886)
458                     {
```

```
                        // Display "OVERLOAD!!".
                        RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
                    }
            else if(in >= 510)
            {
                // clear "overload" display
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n =
100*(.00000120630285*pow(in,3)-.00207707165*pow(in,2)+1.26377865*in-239.473546)
;
            }
            else
            {
                n = 100*(0.04979605*in -0.97505039);
            }
        }

        // +20 V Range
        if (range==2)
        {
            if(in > 753)
            {
                // Display "OVERLOAD!!"
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if(in >= 418)
            {
                // clear "overload" display and turn off overload alarm if on.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n =
100*(.000000459501798*pow(in,3)-.000642990809*pow(in,2)+.326363649*in-49.
6562642);
            }
            else
            {
                n = 100*(.01958838*in - .35450931);
            }
        }


        // +- 60V range
        if (range==3)
        {

            if(in > 822)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
```

```
503             }
504             else if(in >= 644)
505             {
506                 // clear "overload" display.
507                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
508                 n = 100*(.000463716249*pow(in,2)-.513499415*in+173.509826);
509             }
510             else if(in >= 536)
511             {
512                 n =
… 100*(.0000471723712*pow(in,3)-.0834224599*pow(in,2)+49.3036761*n-9708.93574);
513             }
514             else if(in >= 466)
515             {
516                 n =
… 100*(.000138387529*pow(in,3)-.199873792*pow(in,2)+96.3164326*n-15493.7468);
517             }
518             else
519             {
520                 n = 100*(.10634411*n-60.01304714);
521             }
522         }
523 
524 
525         // +- 20V range
526         if (range==4)
527         {
528             if(in > 726)
529             {
530                 // Display "OVERLOAD!!".
531                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
532             }
533             else if(in >= 638)
534             {
535                 // clear "overload" display.
536                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
537                 n = 100*(.000370271737*pow(in,2)-.41544914*in+130.330998);
538             }
539             else if(in >= 562)
540             {
541                 n =
… 100*(.0000270833604*pow(in,3)-.0487960905*pow(in,2)+29.3974651*n-5908.83668);
542             }
543             else if(in >= 503)
544             {
545                 n =
… 100*(.000059160487*pow(in,3)-.0910737114*pow(in,2)+46.7713447*n-8012.48998);
546             }
```

```
            else
            {
                n = 100*(.03929453*n-20.29708651);
            }
        }

        if (range==5)
        {
        // this is used for AC measurement while the bias voltage is applied. It should be able to measure
        // 60 VAC (with no input DC bias).
            if(in > 890)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if (in >= 316)
            {
                // clear "overload" display.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n = 100*(.000000190678554*pow(in,3)-.000247037481*pow(in,2)+.170832813*n-10.5325295);
            }
            else
            {
                n = 100*(.00000240912599*pow(in,3)-.00107040998*pow(in,2)+.21867249*in-2.29882854);
            }
        }

        //20V AC range
        if (range==6)
        {
        // this is used for AC measurement while the bias voltage is applied. It should be able to measure
        // 20 VAC (with no input DC bias).
            if(in > 700)
            {
                // Display "OVERLOAD!!".
                RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
            }
            else if (in > 470)
            {
                // clear "overload" display.
                RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
                n = 100*(.00000105180291*pow(in,3)-.00160346156*pow(in,2)+.848433881*in-139.767537)
```

```
588… ;
589              }
590              else
591              {
592                  n = 100*(0.03259544*in-1.08252444);
593              }
594          }
595      }
596
597      if (channel==4)
598      {
599          // +60V range
600          if (range==1)
601          {
602              if(in > 858)
603                      {
604                          // Display "OVERLOAD!!".
605                          RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
606                      }
607              else if(in >= 505)
608              {
609                  // clear "overload" display
610                  RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
611                  n =
… 100*(.00000127468593*pow(in,3)-.00215988023*pow(in,2)+1.29593845*in-237.930937)
… ;
612              }
613              else
614              {
615                  n = 100*(0.06084352*in -1.06810412);
616              }
617          }
618
619          // +20 V Range
620          if (range==2)
621          {
622              if(in > 756)
623              {
624                  // Display "OVERLOAD!!"
625                  RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
626              }
627              else if(in >= 421)
628              {
629                  // clear "overload" display and turn off overload alarm if on.
630                  RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
631                  n =
… 100*(.000000518158013*pow(in,3)-.000739041249*pow(in,2)+.375546201*in-57.
… 8020384);
```

```
632             }
633             else
634             {
635                 n = 100*(.01929776*in - .32886281);
636             }
637         }
638 
639         // +- 60V range
640         if (range==3)
641         {
642             if(in > 786)
643             {
644                 // Display "OVERLOAD!!".
645                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
646             }
647             else if(in >= 632)
648             {
649                 // clear "overload" display.
650                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
651                 n = 100*(.000417850145*pow(in,2)-.434178446*in+147.549041);
652             }
653             else if(in >= 508)
654             {
655                 n =
… 100*(.0000156085684*pow(in,3)-.0282788058*pow(in,2)+17.1602086*n-3450.64843);
656             }
657             else if(in >= 462)
658             {
659                 n =
… 100*(.000210316*pow(in,3)-.2880891*pow(in,2)+131.483771*n-19999.1093);
660             }
661             else
662             {
663                 n = 100*(.11781646*n-60.01993276);
664             }
665         }
666 
667         // +- 20V range
668         if (range==4)
669         {
670             if(in > 735)
671             {
672                 // Display "OVERLOAD!!".
673                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
674             }
675             else if(in >= 614)
676             {
677                 // clear "overload" display.
```

```
678                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
679                 n = 100*(.000309781869*pow(in,2)-.329670592*in+99.6480572);
680             }
681             else if(in >= 536)
682             {
683                 n =
… 100*(.0000131501929*pow(in,3)-.0234968693*pow(in,2)+14.0607423*n-2805.00816);
684             }
685             else if(in >= 465)
686             {
687                 n =
… 100*(.0000908751659*pow(in,3)-.134398957*pow(in,2)+66.2523055*n-10885.9161);
688             }
689             else
690             {
691                 n = 100*(.03939923*n-20.34388101);
692             }
693         }
694 
695         if (range==5)
696         {
697         // this is used for AC measurement while the bias voltage is applied.
… It should be able to measure
698         // 60 VAC (with no input DC bias).
699             if(in > 917)
700             {
701                 // Display "OVERLOAD!!".
702                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
703             }
704             else if (in >= 830)
705             {
706                 // clear "overload" display.
707                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
708                 n = 100*(.00166286043*pow(in,2)-2.68374256*n+1142.01152);
709             }
710             else if (in >= 311)
711             {
712                 n = 100*(.06774023*in+2.43320481);
713             }
714             else
715             {
716                 n = 100*(-.000133665982*pow(in,2)+.125135292*in-2.14617421);
717             }
718         }
719 
720         //20V AC range
721         if (range==6)
722         {
```

```
723         // this is used for AC measurement while the bias voltage is applied.
  … It should be able to measure
724         // 20 VAC (with no input DC bias).
725             if(in > 720)
726             {
727                 // Display "OVERLOAD!!".
728                 RIT128x96x4StringDraw("OVERLOAD!!", 35, 12, 15);
729             }
730             else if (in > 603)
731             {
732                 // clear "overload" display.
733                 RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
734                 n = 100*(.08866799*in-37.45685885);
735             }
736             else
737             {
738                 n = 100*(0.02808476*in-.36873818);
739             }
740         }
741     }
742 
743         // The number n is the read voltage returned for display.
744         return n;
745     }
746 
747 // This function converts fixed point number to ASCII string
748 // format signed 32-bit with resolution 0.001
749 // range -999.99 to +999.99
750 // Input: signed 32-bit integer part of fixed point number
751 // Output: null-terminated string exactly 7 characters plus null
752 // Examples
753 //  12345 to " 123.45"
754 // -82100 to "-821.00"
755 //   -102 to "  -1.02"
756 //     31 to "   0.31"
757 // Derived from LM3S1968 firmware package, via
  … http://users.ece.utexas.edu/~valvano/arm/rit128x96x4.c
758 void Fix2Str(long const num, char *string){
759   long n;
760 
761   if(num<0){
762     n = -num;
763     string[0] = '-';
764   } else{
765     n = num;
766     string[0] = ' ';
767   }
768   if(n>99999){      // too big
```

```
    string[1] = '*';
    string[2] = '*';
    string[3] = '*';
    string[4] = '.';
    string[5] = '*';
    string[6] = '*';
    string[7] = 0;
    return;
  }  if(n>9999){
    string[1] = '0'+n/10000;
    n = n%10000;
    string[2] = '0'+n/1000;
  } else{
    if(n>999){
      if(num<0){
        string[0] = ' ';
        string[1] = '-';
      } else {
        string[1] = ' ';
      }
      string[2] = '0'+n/1000;
    } else{
      if(num<0){
        string[0] = ' ';
        string[1] = ' ';
        string[2] = '-';
      } else {
        string[1] = ' ';
        string[2] = ' ';
      }
    }
  }
  n = n%1000;
  string[3] = '0'+n/100;
  n = n%100;
  string[4] = '.';
  string[5] = '0'+n/10;
  n = n%10;
  string[6] = '0'+n;
  string[7] = 0;
}



//*************************************************************************
//
//! Displays a fixed-point number (0.01 resolution) on the OLED display.
//!
```

```
817 //! \param num is the integer part of the fixed-point number to display.
818 //! \param ulX is the horizontal position to display the string, specified in
819 //! columns from the left edge of the display.
820 //! \param ulY is the vertical position to display the string, specified in
821 //! rows from the top edge of the display.
822 //! \param ucLevel is the 4-bit gray scale value to be used for displayed text.
823 //!
824 //! This function will display the number on the display.
825 //! The num should be between -99999 and 99999
826 //!
827 //! If the drawing of the string reaches the right edge of the display, no more
828 //! characters will be drawn.  Therefore, special care is not required to avoid
829 //! supplying a string that is ``too long'' to display.
830 //!
831 //! \note Because the OLED display packs 2 pixels of data in a single byte, the
832 //! parameter \e ulX must be an even column number (for example, 0, 2, 4, and
833 //! so on).
834 //!
835 //! \return None.
836 //
837 //*****************************************************************************
838 void
839 RIT128x96x4FixOut2(long num, unsigned long ulX, unsigned long ulY, unsigned
  … char ucLevel)
840 {
841     char string[10];
842     Fix2Str(num, string);
843     RIT128x96x4StringDraw(string, ulX, ulY, ucLevel);
844 }
845
846 /***************UInt2Str4**************
847  converts unsigned integer number to ASCII string
848  format unsigned 32-bit
849  range 0 to 9999
850  Input: unsigned 32-bit integer
851  Output: null-terminated string exactly 4 characters plus null
852  Examples
853   1234 to "1234"
854   821  to " 821"
855   10   to "  10"
856   3    to "   3"
857
858  */
859
860
861 void UInt2Str4(unsigned long const num, char *string){
862   unsigned long n=num;
863
```

```
864
865   if(n>9999){       // too big
866     string[0] = '*';
867     string[1] = '*';
868     string[2] = '*';
869     string[3] = '*';
870     string[4] = 0;
871     return;
872   }
873   if(n>999){   // 1000 to 9999
874     string[0] = '0'+n/1000;
875     n = n%1000;
876     string[1] = '0'+n/100;
877     n = n%100;
878     string[2] = '0'+n/10;
879     n = n%10;
880     string[3] = '0'+n;
881     string[4] = 0;
882     return;
883   }
884   if(n>99){   // 100 to 999
885     string[0] = ' ';
886     string[1] = '0'+n/100;
887     n = n%100;
888     string[2] = '0'+n/10;
889     n = n%10;
890     string[3] = '0'+n;
891     string[4] = 0;
892     return;
893   }
894   if(n>9){   // 10 to 99
895     string[0] = ' ';
896     string[1] = ' ';
897     string[2] = '0'+n/10;
898     n = n%10;
899     string[3] = '0'+n;
900     string[4] = 0;
901     return;
902   }
903   // 0 to 9
904   string[0] = ' ';
905   string[1] = ' ';
906   string[2] = ' ';
907   string[3] = '0'+n;
908   string[4] = 0;
909 }
910
911 //*****************************************************************************
```

```
912 //
913 //! Displays an integer on the OLED display.
914 //!
915 //! \param num is the integer to display.
916 //! \param ulX is the horizontal position to display the string, specified in
917 //! columns from the left edge of the display.
918 //! \param ulY is the vertical position to display the string, specified in
919 //! rows from the top edge of the display.
920 //! \param ucLevel is the 4-bit gray scale value to be used for displayed text.
921 //!
922 //! This function will display the number on the display.
923 //! The num should be between 0 and 9999
924 //!
925 //! If the drawing of the string reaches the right edge of the display, no more
926 //! characters will be drawn.  Therefore, special care is not required to avoid
927 //! supplying a string that is ``too long'' to display.
928 //!
929 //! \note Because the OLED display packs 2 pixels of data in a single byte, the
930 //! parameter \e ulX must be an even column number (for example, 0, 2, 4, and
931 //! so on).
932 //!
933 //! \return None.
934 //
935 //*****************************************************************************
936
937
938 void RIT128x96x4UDecOut4(unsigned long num, unsigned long ulX, unsigned long
  … ulY, unsigned char ucLevel)
939 {
940     char string[10];
941   UInt2Str4(num, string);
942   RIT128x96x4StringDraw(string, ulX, ulY, ucLevel);
943 }
944
945 // This is a FIFO buffer which stores the last 80 samples
946 // for channel 1 and returns the sum of the buffer.
947 long DCAvg1(long value1)
948 {
949         if(i1==79)
950         {
951             ADCValueAvg1[i1]=value1;
952             Sum1 = Sum1 + ADCValueAvg1[i1]-ADCValueAvg1[0];
953             i1=0;
954         }
955         else
956         {
957             ADCValueAvg1[i1]=value1;
958             Sum1 = Sum1 + ADCValueAvg1[i1]-ADCValueAvg1[i1+1];
```

```
            i1++;
        }
    return Sum1;
}

// This is a FIFO buffer which stores the last 80 samples
// for channel 2 and returns the sum of the buffer.
long DCAvg2(long value2)
{
        if(i2==79)
        {
            ADCValueAvg2[i2]=value2;
            Sum2 = Sum2 + ADCValueAvg2[i2]-ADCValueAvg2[0];
            i2=0;
        }
        else
        {
            ADCValueAvg2[i2]=value2;
            Sum2 = Sum2 + ADCValueAvg2[i2]-ADCValueAvg2[i2+1];
            i2++;
        }
    return Sum2;
}

// This is a FIFO buffer which stores the last 80 samples
// for channel 3 and returns the sum of the buffer.
long DCAvg3(long value3)
{
        if(i3==79)
        {
            ADCValueAvg3[i3]=value3;
            Sum3 = Sum3 + ADCValueAvg3[i3]-ADCValueAvg3[0];
            i3=0;
        }
        else
        {
            ADCValueAvg3[i3]=value3;
            Sum3 = Sum3 + ADCValueAvg3[i3]-ADCValueAvg3[i3+1];
            i3++;
        }
    return Sum3;
}

// This is a FIFO buffer which stores the last 80 samples
// for channel 4 and returns the sum of the buffer.
long DCAvg4(long value4)
{
        if(i4==79)
```

```
        {
            ADCValueAvg4[i4]=value4;
            Sum4 = Sum4 + ADCValueAvg4[i4]-ADCValueAvg4[0];
            i4=0;
        }
        else
        {
            ADCValueAvg4[i4]=value4;
            Sum4 = Sum4 + ADCValueAvg4[i4]-ADCValueAvg4[i4+1];
            i4++;
        }
    return Sum4;
}

// Decides when Channel 1 should switch out of Range 1
void Ch1Range1SW()
{
    if(i1_1==239)
    {
        i1_1=0;
        i1_c1 = 0;
        i1_c2 = 0;
    }
    else if(ch1ADCDisplay < 5)
    {
        i1_c1++;
        if(i1_c1 >= 120)
        {
            range_1 = 3;
            // turning on the +-60 Mosfet gates for channel 1 channels
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        }
    }
    else if(ch1ADCDisplay < 365)
    {
        i1_c2++;
        if(i1_c2 >= 120)
        {
            range_1 = 2;
            // turning off all Mosfet gates for channel 1 +20
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
        }
    }
    if(i1_1!=159)
```

```
    {
        i1_1++;
    }
}

// Decides when Channel 1 should switch out of Range 2
void Ch1Range2SW()
{
    if(i1_2==239)
    {
        i1_2=0;
        i1_c3 = 0;
        i1_c4 = 0;
    }
    if(ch1ADCDisplay < 5)
    {
        i1_c3++;
        if(i1_c3 >= 120)
        {
            range_1 = 3;
            // turning on the +-60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        }
    }
    if(ch1ADCDisplay > 748)
    {
        i1_c4++;
        if(i1_c4 >= 120)
        {
            range_1 = 1;
            // turning on the +60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
        }
    }
    if(i1_2!=159)
    {
        i1_2++;
    }
}

// Decides when Channel 1 should switch out of Range 3
void Ch1Range3SW()
{
```

```
    if(i1_3==239)
    {
        i1_3=0;
        i1_c5 = 0;
        i1_c6 = 0;
    }
    if(ch1ADCDisplay > 513)
    {
        i1_c5++;
        if(i1_c5 >= 120)
        {
            range_1 = 1;
            // turning on the +60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);        }
    }
    if((minimum1 > 354)&&(maximum1 < 544))
    {
        i1_c6++;
        if(i1_c6 >= 120)
        {
            range_1 = 4;
            // turning on the +-20 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
        }
    }
    if(i1_3!=159)
    {
        i1_3++;
    }
}

// Decides when Channel 1 should switch out of Range 4
void Ch1Range4SW()
{
    if(i1_4==239)
    {
        i1_4=0;
        i1_c7 = 0;
        i1_c8 = 0;
    }
    if(ch1ADCDisplay > 522)
    {
        i1_c7++;
```

```
        if(i1_c7 >= 120)
        {
            range_1 = 1;
            // turning on the +60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
        }
    }
    if(ch1ADCDisplay < 3)
    {
        i1_c8++;
        if(i1_c8 >= 120)
        {
            range_1 = 3;
            // turning on the +-60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        }
    }
    if(i1_4!=159)
    {
        i1_4++;
    }
}

// Decides when Channel 1 should switch out of Range 5
void Ch1Range5SW()
{
    if(i1_5==239)
    {
        i1_5=0;
        i1_c9 = 0;
    }

    else if((maximum1-minimum1) < 277)
    {
        i1_c9++;
        if(i1_c9 >= 120)
        {
            range_1 = 6;
            // turning on the +-20 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
```

```
        }
    }
    if(i1_5!=159)
    {
        i1_5++;
    }
}

// Decides when Channel 1 should switch out of Range 6
void Ch1Range6SW()
{
    if(i1_6==239)
    {
        i1_6=0;
        i1_c10 = 0;
    }
    if((maximum1-minimum1) > 678)
    {
        i1_c10++;
        if(i1_c10 >= 120)
        {
            range_1 = 5;
            // turning on the +-60 Mosfet gates for channel 1
            GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        }
    }
    if(i1_6!=159)
    {
        i1_6++;
    }
}

// Decides when Channel 2 should switch out of Range 1
void Ch2Range1SW()
{
    if(i2_1==239)
    {
        i2_1=0;
        i2_c1 = 0;
        i2_c2 = 0;
    }
    else if(ch2ADCDisplay < 5)
    {
        i2_c1++;
        if(i2_c1 >= 120)
```

```
        {
            range_2 = 3;
            // turning on the +-60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7,
GPIO_PIN_4|GPIO_PIN_7);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
        }
    }
    else if(ch2ADCDisplay < 408)
    {
        i2_c2++;
        if(i2_c2 >= 120)
        {
            range_2 = 2;
            // turning off all Mosfet gates for channel 2 +20
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7, 0);
        }
    }
    if(i2_1!=159)
    {
        i2_1++;
    }
}

// Decides when Channel 2 should switch out of Range 2
void Ch2Range2SW()
{
    if(i2_2==239)
    {
        i2_2=0;
        i2_c3 = 0;
        i2_c4 = 0;
    }
    if(ch2ADCDisplay <= 14)
    {
        i2_c3++;
        if(i2_c3 >= 120)
        {
            range_2 = 3;
            // turning on the +-60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7,
GPIO_PIN_4|GPIO_PIN_7);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
        }
    }
```

```
    if(ch2ADCDisplay > 704)
    {
        i2_c4++;
        if(i2_c4 >= 120)
        {
            range_2 = 1;
            // turning on the +60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_7, 0);
        }
    }
    if(i2_2!=159)
    {
        i2_2++;
    }
}

// Decides when Channel 2 should switch out of Range 3
void Ch2Range3SW()
{
    if(i2_3==239)
    {
        i2_3=0;
        i2_c5 = 0;
        i2_c6 = 0;
    }
    if(ch2ADCDisplay > 535)
    {
        i2_c5++;
        if(i2_c5 >= 120)
        {
            range_2 = 1;
            // turning on the +60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_7, 0);
        }
    }
    if((minimum2 > 368)&&(maximum2 < 575))
    {
        i2_c6++;
        if(i2_c6 >= 120)
        {
            range_2 = 4;
            // turning on the +-20 Mosfet gates for channel 2
```

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7, 0);
        }
    }
    if(i2_3!=159)
    {
        i2_3++;
    }
}

// Decides when Channel 2 should switch out of Range 4
void Ch2Range4SW()
{
    if(i2_4==239)
    {
        i2_4=0;
        i2_c7 = 0;
        i2_c8 = 0;
    }
    if(ch2ADCDisplay > 512)
    {
        i2_c7++;
        if(i2_c7 >= 120)
        {
            range_2 = 1;
            // turning on the +60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_7, 0);
        }
    }
    if(ch2ADCDisplay < 5)
    {
        i2_c8++;
        if(i2_c8 >= 120)
        {
            range_2 = 3;
            // turning on the +-60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7,
GPIO_PIN_4|GPIO_PIN_7);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
        }
    }
    if(i2_4!=159)
    {
```

```
        i2_4++;
    }
}

// Decides when Channel 2 should switch out of Range 5
void Ch2Range5SW()
{
    if(i2_5==239)
    {
        i2_5=0;
        i2_c9 = 0;
    }

    else if((maximum2-minimum2) < 233)
    {
        i2_c9++;
        if(i2_c9 >= 120)
        {
            range_2 = 6;
            // turning on the +-20 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7, 0);
        }
    }
    if(i2_5!=159)
    {
        i2_5++;
    }
}

// Decides when Channel 2 should switch out of Range 6
void Ch2Range6SW()
{
    if(i2_6==239)
    {
        i2_6=0;
        i2_c10 = 0;
    }
    if((maximum2-minimum2) > 597)
    {
        i2_c10++;
        if(i2_c10 >= 120)
        {
            range_2 = 5;
            // turning on the +-60 Mosfet gates for channel 2
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7,
GPIO_PIN_4|GPIO_PIN_7);
```

```
            // turning off the other MOSFET gate for channel 2
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
        }
    }
    if(i2_6!=159)
    {
        i2_6++;
    }
}

// Decides when Channel 3 should switch out of Range 1
void Ch3Range1SW()
{
    if(i3_1==239)
    {
        i3_1=0;
        i3_c1 = 0;
        i3_c2 = 0;
    }
    else if(ch3ADCDisplay < 5)
    {
        i3_c1++;
        if(i3_c1 >= 120)
        {
            range_3 = 3;
            // turning on the +-60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    else if(ch3ADCDisplay < 420)
    {
        i3_c2++;
        if(i3_c2 >= 120)
        {
            range_3 = 2;
            // turning off all Mosfet gates for channel 3 +20
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(i3_1!=159)
    {
        i3_1++;
    }
```

```
}

// Decides when Channel 3 should switch out of Range 2
void Ch3Range2SW()
{
    if(i3_2==239)
    {
        i3_2=0;
        i3_c3 = 0;
        i3_c4 = 0;
    }
    if(ch3ADCDisplay <= 17)
    {
        i3_c3++;
        if(i3_c3 >= 120)
        {
            range_3 = 3;
            // turning on the +-60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(ch3ADCDisplay > 753)
    {
        i3_c4++;
        if(i3_c4 >= 120)
        {
            range_3 = 1;
            // turning on the +60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(i3_2!=159)
    {
        i3_2++;
    }
}

// Decides when Channel 3 should switch out of Range 3
void Ch3Range3SW()
{
    if(i3_3==239)
    {
```

```
        i3_3=0;
        i3_c5 = 0;
        i3_c6 = 0;
    }
    if(ch3ADCDisplay > 600)
    {
        i3_c5++;
        if(i3_c5 >= 120)
        {
            range_3 = 1;
            // turning on the +60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if((minimum3 > 376)&&(maximum3 < 547))
    {
        i3_c6++;
        if(i3_c6 >= 120)
        {
            range_3 = 4;
            // turning on the +-20 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
        }
    }
    if(i3_3!=159)
    {
        i3_3++;
    }
}

// Decides when Channel 3 should switch out of Range 4
void Ch3Range4SW()
{
    if(i3_4==239)
    {
        i3_4=0;
        i3_c7 = 0;
        i3_c8 = 0;
    }
    if(ch3ADCDisplay > 503)
    {
        i3_c7++;
```

```
        if(i3_c7 >= 120)
        {
            range_3 = 1;
            // turning on the +60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(ch3ADCDisplay < 9)
    {
        i3_c8++;
        if(i3_c8 >= 120)
        {
            range_3 = 3;
            // turning on the +-60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(i3_4!=159)
    {
        i3_4++;
    }
}

// Decides when Channel 3 should switch out of Range 5
void Ch3Range5SW()
{
    if(i3_5==239)
    {
        i3_5=0;
        i3_c9 = 0;
    }
    else if((maximum3-minimum3) < 224)
    {
        i3_c9++;
        if(i3_c9 >= 120)
        {
            range_3 = 6;
            // turning on the +-20 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
```

```
        }
    }
    if(i3_5!=159)
    {
        i3_5++;
    }
}

// Decides when Channel 3 should switch out of Range 6
void Ch3Range6SW()
{
    if(i3_6==239)
    {
        i3_6=0;
        i3_c10 = 0;
    }
    if((maximum3-minimum3) > 614)
    {
        i3_c10++;
        if(i3_c10 >= 120)
        {
            range_3 = 5;
            // turning on the +-60 Mosfet gates for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1);
            // turning off the other MOSFET gate for channel 3
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
        }
    }
    if(i3_6!=159)
    {
        i3_6++;
    }
}

// Decides when Channel 4 should switch out of Range 1
void Ch4Range1SW()
{
    if(i4_1==239)
    {
        i4_1=0;
        i4_c1 = 0;
        i4_c2 = 0;
    }
    else if(ch4ADCDisplay < 5)
    {
        i4_c1++;
        if(i4_c1 >= 120)
```

```
        {
            range_4 = 3;
            // turning on the +-60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6,
GPIO_PIN_4|GPIO_PIN_6);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
        }
    }
    else if(ch4ADCDisplay < 346)
    {
        i4_c2++;
        if(i4_c2 >= 120)
        {
            range_4 = 2;
            // turning off all Mosfet gates for channel 4 +20
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6, 0);
        }
    }
    if(i4_1!=159)
    {
        i4_1++;
    }
}

// Decides when Channel 4 should switch out of Range 2
void Ch4Range2SW()
{
    if(i4_2==239)
    {
        i4_2=0;
        i4_c3 = 0;
        i4_c4 = 0;
    }
    if(ch4ADCDisplay <= 18)
    {
        i4_c3++;
        if(i4_c3 >= 120)
        {
            range_4 = 3;
            // turning on the +-60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6,
GPIO_PIN_4|GPIO_PIN_6);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
        }
    }
    if(ch4ADCDisplay > 694)
```

```
    {
        i4_c4++;
        if(i4_c4 >= 120)
        {
            range_4 = 1;
            // turning on the +60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5|GPIO_PIN_6, 0);
        }
    }
    if(i4_2!=159)
    {
        i4_2++;
    }
}

// Decides when Channel 4 should switch out of Range 3
void Ch4Range3SW()
{
    if(i4_3==239)
    {
        i4_3=0;
        i4_c5 = 0;
        i4_c6 = 0;
    }
    if(ch4ADCDisplay > 525)
    {
        i4_c5++;
        if(i4_c5 >= 120)
        {
            range_4 = 1;
            // turning on the +60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5|GPIO_PIN_6, 0);
        }
    }
    if((minimum4 > 341)&&(maximum4 < 517))
    {
        i4_c6++;
        if(i4_c6 >= 120)
        {
            range_4 = 4;
            // turning on the +-20 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6, 0);
```

```
        }
    }
    if(i4_3!=159)
    {
        i4_3++;
    }
}

// Decides when Channel 4 should switch out of Range 4
void Ch4Range4SW()
{
    if(i4_4==239)
    {
        i4_4=0;
        i4_c7 = 0;
        i4_c8 = 0;
    }
    if(ch4ADCDisplay > 505)
    {
        i4_c7++;
        if(i4_c7 >= 120)
        {
            range_4 = 1;
            // turning on the +60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5|GPIO_PIN_6, 0);
        }
    }
    if(ch4ADCDisplay < 8)
    {
        i4_c8++;
        if(i4_c8 >= 120)
        {
            range_4 = 3;
            // turning on the +-60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6,
GPIO_PIN_4|GPIO_PIN_6);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
        }
    }
    if(i4_4!=159)
    {
        i4_4++;
    }
}

```

```
// Decides when Channel 4 should switch out of Range 5
void Ch4Range5SW()
{
    if(i4_5==239)
    {
        i4_5=0;
        i4_c9 = 0;
    }

    else if((maximum4-minimum4) < 234)
    {
        i4_c9++;
        if(i4_c9 >= 120)
        {
            range_4 = 6;
            // turning on the +-20 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6, 0);
        }
    }
    if(i4_5!=159)
    {
        i4_5++;
    }
}

// Decides when Channel 4 should switch out of Range 6
void Ch4Range6SW()
{
    if(i4_6==239)
    {
        i4_6=0;
        i4_c10 = 0;
    }
    if((maximum4-minimum4) > 642)
    {
        i4_c10++;
        if(i4_c10 >= 120)
        {
            range_4 = 5;
            // turning on the +-60 Mosfet gates for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6,
GPIO_PIN_4|GPIO_PIN_6);
            // turning off the other MOSFET gate for channel 4
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
        }
    }
```

```
    if(i4_6!=159)
    {
        i4_6++;
    }
}

int
main(void)
{
  // This array is used for storing the data read from the ADC FIFO. It
  // must be as large as the FIFO for the sequencer in use.  This
  // uses sequence 2 which has a FIFO depth of 4.
  unsigned long ulADC0_Value[4];

  // Set the clocking to run at 20 MHz (200 MHz / 10) using the PLL.
  SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_8MHZ);

  // Initialize the OLED display.
  RIT128x96x4Init(1000000);

  // Enable processor interrupts.
  IntMasterEnable();

    // Enable the peripherals.

  // GPIO E used for "Up" button
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    // GPIO B, D, F, and G used for MOSFET switching
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);

  // ADC0 being enabled
  SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    // Configure the "up" "down" "left" and "right" buttons.
  GPIOPinTypeGPIOInput(GPIO_PORTE_BASE,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
  GPIOPadConfigSet(GPIO_PORTE_BASE,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPU);

    //Configuring pins used for MOSFET gates.
    //G0=PG0
    GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_0);
```

```
1909      // D2=U1Rx, D3=U1Tx, D4=CCP0, D5=CCP1, D7=IDX0,
1910      GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,
… GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_7);
1911      //F1=IDX1 F3=LED0
1912    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
1913      //  B1=PWM3, B4=CO-, B5=C1-, B6=CO+
1914      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,
… GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
1915
1916    // turning on the +60 Mosfet gates for all 4 channels
1917      GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
1918      GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
1919      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
1920      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4);
1921
1922      //turning off the 20 and 60 bias gates for all 4 channels
1923      GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_7, 0);
1924      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
1925      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1|GPIO_PIN_5|GPIO_PIN_6, 0);
1926      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
1927
1928    // Enable sample sequence 2 with a processor signal trigger.  Sequence 2
1929    // will do a single sample when the processor sends a signal to start the
1930    // conversion.  Each ADC module has 4 programmable sequences, sequence 0
1931    // to sequence 2.  This is using sequence 2.
1932    //
1933    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 2);
1934
1935    //
1936    // Configure step 0 on sequence 3.  Sample channel 0 (ADC_CTL_CH0) in
1937    // single-ended mode (default). Configure step 1 for channel 1,
1938      // step 2 for channel 2, and step 4 for channel 3. Configure the interrupt
… flag
1939    // (ADC_CTL_IE) to be set when the sample is done.  Tell the ADC logic
1940    // that this is the last conversion on sequence 2 (ADC_CTL_END).  Sequence
1941    // 2 has four programmable steps.
1942      ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH0);
1943      ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_CH1);
1944      ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH2);
1945      ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH3 | ADC_CTL_IE |
… ADC_CTL_END);
1946
1947    //Enable hardware over-sampling to a factor of 6.
1948    ADCHardwareOversampleConfigure(ADC0_BASE,6);
1949
1950    // Since sample sequence 2 is now configured, it must be enabled.
1951    ADCSequenceEnable(ADC0_BASE, 2);
1952
```

```
1953      // Clear the interrupt status flag.  This is done to make sure the
1954    // interrupt flag is cleared before we sample.
1955    ADCIntClear(ADC0_BASE, 2);
1956
1957    // Drawing labels on display.
1958      RIT128x96x4StringDraw("CH1            CH2", 1, 12, 15);
1959      RIT128x96x4StringDraw("R:1", 1, 20, 15);
1960      RIT128x96x4StringDraw("R:1", 93, 20, 15);
1961      RIT128x96x4StringDraw("A:", 1, 32, 15);
1962      RIT128x96x4StringDraw("V:", 1, 42, 15);
1963      RIT128x96x4StringDraw("CH3            CH4", 1, 52, 15);
1964      RIT128x96x4StringDraw("R:1", 1, 62, 15);
1965      RIT128x96x4StringDraw("R:1", 93, 62, 15);
1966      RIT128x96x4StringDraw("A:", 1, 72, 15);
1967      RIT128x96x4StringDraw("V:", 1, 82, 15);
1968
1969    while(1)
1970    {
1971      // Trigger the ADC conversion.
1972          ADCProcessorTrigger(ADC0_BASE, 2);
1973      // Wait for conversion to be completed.
1974      while(!ADCIntStatus(ADC0_BASE, 2, false))
1975          {
1976          }
1977      // Read ADC Value.
1978      ADCSequenceDataGet(ADC0_BASE, 2, ulADC0_Value);
1979
1980          // If "up" button pressed, toggle AC or DC for channel 1
1981          if(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_0) == 0)
1982          {
1983              if ((range_1==1)|(range_1==2)|(range_1==3)|(range_1==4))
1984              {
1985                  range_1 = 5;
1986                  // turning on the +-60 Mosfet gates for channel 1
1987                  GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
1988                  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
1989                  // turning off the other MOSFET gate for channel 1
1990                  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
1991              }
1992              else
1993              {
1994                  range_1 = 1;
1995                  // turning on the +60 Mosfet gates for channel 1
1996                  GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
1997                  // turning off the other MOSFET gate for channel 1
1998                  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
1999              }
2000          }
```

```
2001
2002        // If "left" button pressed, toggle AC or DC for channel 2
2003        if(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_2) == 0)
2004        {
2005            if ((range_2==1)|(range_2==2)|(range_2==3)|(range_2==4))
2006            {
2007                range_2 = 5;
2008            // turning on the +-60 Mosfet gates for channel 2
2009            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4|GPIO_PIN_7,
  … GPIO_PIN_4|GPIO_PIN_7);
2010            // turning off the other MOSFET gate for channel 2
2011            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
2012            }
2013            else
2014            {
2015                range_2 = 1;
2016                // turning on the +60 Mosfet gates for channel 2
2017                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
2018                // turning off the other MOSFET gate for channel 2
2019                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
2020                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_7, 0);
2021            }
2022        }
2023
2024        // If "down" button pressed, toggle AC or DC for channel 3
2025        if(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_1) == 0)
2026        {
2027            if ((range_3==1)|(range_3==2)|(range_3==3)|(range_3==4))
2028            {
2029                range_3 = 5;
2030            // turning on the +-60 Mosfet gates for channel 3
2031            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
2032            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1);
2033            // turning off the other MOSFET gate for channel 3
2034            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
2035            }
2036            else
2037            {
2038                range_3 = 1;
2039            // turning on the +60 Mosfet gates for channel 3
2040            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
2041            // turning off the other MOSFET gate for channel 3
2042            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
2043            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
2044            }
2045        }
2046
2047
```

```
        // If "right" button pressed, toggle AC or DC for channel 4
        if(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_3) == 0)
        {
            if ((range_4==1)|(range_4==2)|(range_4==3)|(range_4==4))
            {
                range_4 = 5;
                // turning on the +-60 Mosfet gates for channel 4
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4|GPIO_PIN_6,
GPIO_PIN_4|GPIO_PIN_6);
                // turning off the other MOSFET gate for channel 4
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
            }
            else
            {
                range_4 = 1;
                // turning on the +60 Mosfet gates for channel 4
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4);
                // turning off the other MOSFET gate for channel 4
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5|GPIO_PIN_6, 0);
            }
        }

    // Stores the average of the FIFOs for each channel
    ch1ADCDisplay = (DCAvg1(ulADC0_Value[0]) / 80);
    ch2ADCDisplay = (DCAvg2(ulADC0_Value[1]) / 80);
    ch3ADCDisplay = (DCAvg3(ulADC0_Value[2]) / 80);
    ch4ADCDisplay = (DCAvg4(ulADC0_Value[3]) / 80);

    // calculates the maximum and minimum in each FIFO.
    maximum1 = ADCValueAvg1[0];
    minimum1 = ADCValueAvg1[0];
    for (imax1 = 0; imax1 < 80; imax1++)
    {
        if (ADCValueAvg1[imax1] > maximum1)
        {
            maximum1 = ADCValueAvg1[imax1];
        }
        if (ADCValueAvg1[imax1] < minimum1)
        {
            minimum1 = ADCValueAvg1[imax1];
        }
    }
    maximum2 = ADCValueAvg2[0];
    minimum2 = ADCValueAvg2[0];
    for (imax2 = 0; imax2 < 80; imax2++)
    {
        if (ADCValueAvg2[imax2] > maximum2)
        {
```

```
            maximum2 = ADCValueAvg2[imax2];

        }
        if (ADCValueAvg2[imax2] < minimum2)
        {
            minimum2 = ADCValueAvg2[imax2];
        }
    }
    maximum3 = ADCValueAvg3[0];
    minimum3 = ADCValueAvg3[0];
    for (imax3 = 0; imax3 < 80; imax3++)
    {
        if (ADCValueAvg3[imax3] > maximum3)
        {
            maximum3 = ADCValueAvg3[imax3];
        }
        if (ADCValueAvg3[imax3] < minimum3)
        {
            minimum3 = ADCValueAvg3[imax3];
        }
    }
    maximum4 = ADCValueAvg4[0];
    minimum4 = ADCValueAvg4[0];
    for (imax4 = 0; imax4 < 80; imax4++)
    {
        if (ADCValueAvg4[imax4] > maximum4)
        {
            maximum4 = ADCValueAvg4[imax4];
        }
        if (ADCValueAvg4[imax4] < minimum4)
        {
            minimum4 = ADCValueAvg4[imax4];
        }
    }

    // Used for auto-ranging functions.
    if (range_1 == 1)
    {
        RIT128x96x4StringDraw("R:1 DC", 1, 20, 15);
        Ch1Range1SW();
    }
    if (range_1 == 2)
    {
        RIT128x96x4StringDraw("R:2 DC", 1, 20, 15);
        Ch1Range2SW();
    }
    if (range_1 == 3)
    {
```

```
        RIT128x96x4StringDraw("R:3 DC", 1, 20, 15);
        Ch1Range3SW();
    }
    if (range_1 == 4)
    {
        RIT128x96x4StringDraw("R:4 DC", 1, 20, 15);
        Ch1Range4SW();
    }
    if (range_1 == 5)
    {
        RIT128x96x4StringDraw("R:5 AC", 1, 20, 15);
        Ch1Range5SW();
    }
    if (range_1 == 6)
    {
        RIT128x96x4StringDraw("R:6 AC", 1, 20, 15);
        Ch1Range6SW();
    }
    if (range_2 == 1)
    {
        RIT128x96x4StringDraw("R:1 DC", 93, 20, 15);
        Ch2Range1SW();
    }
    if (range_2 == 2)
    {
        RIT128x96x4StringDraw("R:2 DC", 93, 20, 15);
        Ch2Range2SW();
    }
    if (range_2 == 3)
    {
        RIT128x96x4StringDraw("R:3 DC", 93, 20, 15);
        Ch2Range3SW();
    }
    if (range_2 == 4)
    {
        RIT128x96x4StringDraw("R:4 DC", 93, 20, 15);
        Ch2Range4SW();
    }
    if (range_2 == 5)
    {
        RIT128x96x4StringDraw("R:5 AC", 93, 20, 15);
        Ch2Range5SW();
    }
    if (range_2 == 6)
    {
        RIT128x96x4StringDraw("R:6 AC", 93, 20, 15);
        Ch2Range6SW();
    }
```

```
    if (range_3 == 1)
    {
        RIT128x96x4StringDraw("R:1 DC", 1, 62, 15);
        Ch3Range1SW();
    }
    if (range_3 == 2)
    {
        RIT128x96x4StringDraw("R:2 DC", 1, 62, 15);
        Ch3Range2SW();
    }
    if (range_3 == 3)
    {
        RIT128x96x4StringDraw("R:3 DC", 1, 62, 15);
        Ch3Range3SW();
    }
    if (range_3 == 4)
    {
        RIT128x96x4StringDraw("R:4 DC", 1, 62, 15);
        Ch3Range4SW();
    }
    if (range_3 == 5)
    {
        RIT128x96x4StringDraw("R:5 AC", 1, 62, 15);
        Ch3Range5SW();
    }
    if (range_3 == 6)
    {
        RIT128x96x4StringDraw("R:6 AC", 1, 62, 15);
        Ch3Range6SW();
    }
    if (range_4 == 1)
    {
        RIT128x96x4StringDraw("R:1 DC", 93, 62, 15);
        Ch4Range1SW();
    }
    if (range_4 == 2)
    {
        RIT128x96x4StringDraw("R:2 DC", 93, 62, 15);
        Ch4Range2SW();
    }
    if (range_4 == 3)
    {
        RIT128x96x4StringDraw("R:3 DC", 93, 62, 15);
        Ch4Range3SW();
    }
    if (range_4 == 4)
    {
        RIT128x96x4StringDraw("R:4 DC", 93, 62, 15);
```

```
2239            Ch4Range4SW();
2240        }
2241        if (range_4 == 5)
2242        {
2243            RIT128x96x4StringDraw("R:5 AC", 93, 62, 15);
2244            Ch4Range5SW();
2245        }
2246        if (range_4 == 6)
2247        {
2248            RIT128x96x4StringDraw("R:6 AC", 93, 62, 15);
2249            Ch4Range6SW();
2250        }
2251
2252        // Displaying values on OLED. displayRate is set here.
2253        if(displayRate==7)
2254        {
2255            RIT128x96x4StringDraw("A:", 65, 32, 15);
2256            RIT128x96x4StringDraw("V:", 65, 42, 15);
2257            RIT128x96x4StringDraw("A:", 1, 72, 15);
2258            RIT128x96x4StringDraw("V:", 1, 82, 15);
2259            // If channel 1 set for AC measurement, display proper AC values.
2260            if(range_1==5|range_1==6)
2261            {
2262                RIT128x96x4UDecOut4(maximum1-minimum1, 30, 32, 15);
2263                RIT128x96x4FixOut2(adj_n((maximum1 - minimum1), range_1, 1), 15,
    42, 15);
2264            }
2265            // Otherwise, display DC values.
2266            else
2267            {
2268                RIT128x96x4UDecOut4(ch1ADCDisplay, 30, 32, 15);
2269                RIT128x96x4FixOut2(adj_n(ch1ADCDisplay, range_1, 1), 15, 42, 15);
2270            }
2271            // if channel 2 set for AC measurement, display proper AC values.
2272            if(range_2==5|range_2==6)
2273            {
2274                RIT128x96x4UDecOut4(maximum2-minimum2, 90, 32, 15);
2275                RIT128x96x4FixOut2(adj_n((maximum2 - minimum2), range_2, 2), 75,
    42, 15);
2276            }
2277            // Otherwise, display DC values.
2278            else
2279            {
2280                RIT128x96x4UDecOut4(ch2ADCDisplay, 90, 32, 15);
2281                RIT128x96x4FixOut2(adj_n(ch2ADCDisplay, range_2, 2), 75, 42, 15);
2282            }
2283            // if channel 3 set for AC measurement, display proper AC values.
2284            if(range_3==5|range_3==6)
```

```
2285            {
2286                RIT128x96x4UDecOut4((maximum3-minimum3), 30, 72, 15);
2287                RIT128x96x4FixOut2(adj_n((maximum3-minimum3), range_3, 3), 15, 82,
    … 15);
2288            }
2289            // Otherwise, display DC values.
2290            else
2291            {
2292                RIT128x96x4UDecOut4(ch3ADCDisplay, 30, 72, 15);
2293                RIT128x96x4FixOut2(adj_n(ch3ADCDisplay, range_3, 3), 15, 82, 15);
2294            }
2295            // if channel 4 set for AC measurement, display proper AC values.
2296            if(range_4==5|range_4==6)
2297            {
2298                RIT128x96x4UDecOut4((maximum4-minimum4), 90, 72, 15);
2299                RIT128x96x4FixOut2(adj_n((maximum4-minimum4), range_4, 4), 75, 82,
    … 15);
2300            }
2301            // Otherwise, display DC values.
2302            else
2303            {
2304                RIT128x96x4UDecOut4(ch4ADCDisplay, 90, 72, 15);
2305                RIT128x96x4FixOut2(adj_n(ch4ADCDisplay, range_4, 4), 75, 82, 15);
2306            }
2307            displayRate=0;
2308        }
2309        else displayRate++;
2310        ADCIntClear(ADC0_BASE, 2);
2311
2312        //
2313        // This function provides a means of generating a constant length
2314        // delay.  The function delay (in cycles) = 3 * parameter.  Delay
2315        // 40ms arbitrarily.
2316        //
2317        SysCtlDelay(SysCtlClockGet()/75);
2318      }
2319    }
2320
```

BIBLIOGRAPHY

[1] “Tektronix and Keithley Digital Multimeters price list”, http://www.tek.com/digital-multimeter/high-resolution-digital-multimeters

[2] “Extech Instruments MN35 User Manual”, http://extechvietnam.com/hinh-anh/files/MN35_UM.pdf

[3] “Stellaris® LM3S8962 Evaluation Board User’s Manual”, http://users.ece.utexas.edu/~valvano/EE345M/LM3S8962EvalBoard.pdf

[4] “Digital multimeter shield for Arduino”, http://www.instructables.com/id/Digital-multimeter-shield-for-Arduino

[5] “Stellaris® LM3S8962 Microcontroller data sheet”, http://www.ti.com/lit/ds/spms001g/spms001g.pdf

[6] “Diodes, Inc. ZVN4424A datasheet”, http://www.diodes.com/_files/datasheets/ZVN4424A.pdf

[7] “Fluke 8060a True-rms Multimeter Instruction Manual”, http://assets.fluke.com/manuals/8060a_3vimeng0200.pdf

[8] “rit128x96x4.c code for the LM3S1968 evaluation board”, http://users.ece.utexas.edu/~valvano/arm/rit128x96x4.c